Bilkent University

Department of Computer Engineering

Senior Design Project

T2426 - CheckMate

Design Project Final Report

Ayberk Berat Eroğlu - 22103675
Alp Batu Aksan - 22103246
Pelin Öner - 22102409
İpek Sönmez - 22103939
Efe Tuna Can - 22102127

Supervisor: Cevdet Aykanat

1. Introduction

In an era characterized by the rapid dissemination of misinformation, ensuring the accuracy and reliability of online content has become a critical challenge. Fact-checking remains predominantly a manual process, relying on human experts to investigate claims and verify their authenticity. For instance, Meta, the parent company of Facebook and Instagram, employs human fact-checkers to assess the credibility of online content [1]. Similarly, Twitter (now X) utilizes Community Notes, a crowdsourced system that aggregates contextual insights and fact-checks based on user contributions [2]. However, these approaches are often insufficient to address the vast volume of information circulating on digital platforms, as manual fact-checking is both time-intensive and resource-demanding [3].

CheckMate, an artificial intelligence (AI) powered browser extension and mobile application designed to automate fact-checking, multimedia verification, political bias detection, and subjectivity analysis. By leveraging machine learning (ML) techniques, CheckMate aims to provide a more efficient and scalable solution for misinformation detection. As the internet continues to emerge as the dominant news source surpassing traditional media in countries such as the United Kingdom [4] the integration of automated verification tools across major news platforms is essential for mitigating the spread of false information.

CheckMate systematically evaluates news articles and assigns a reliability score based on multiple factors. Using natural language processing (NLP), it detects subjective language and analyzes political bias, incorporating these findings into the reliability score assessment. Additionally, CheckMate cross-references information with other sources via search Application Programming Interfaces (APIs) to identify confirmatory or contradictory evidence.

The publication source is also analyzed to assess credibility. If a source has a history of publishing misinformation, exhibits political bias, or consistently employs subjective language, CheckMate adjusts the source's credibility score accordingly, which contributes to the overall reliability score of the article. Additionally, CheckMate evaluates the objectivity of article titles and performs grammar analysis to identify linguistic indicators of reliability or potential manipulation.

Upon completing its analysis, CheckMate presents the reliability score along with a rationale, highlighting factors such as high political bias, subjective language, poor grammar quality, subjective titling, or contradictions with reputable sources.

2. Requirements Details

CheckMate allows users to submit news articles, perform credibility and bias analysis, view detailed reliability reports, manage user accounts, and track their analysis history based on subscription level.

2.1 Functional Requirements

The CheckMate system is designed with the following specific functional capabilities:

- CheckMate maintains a reference database of verified news sources regularly updated to incorporate emerging and reliable sources.
- The reference database is updated periodically to include newly emerging and credible sources.
- CheckMate administrators have the capability to update and manage the reference database manually or via automated scripts.
- The system compares user-submitted news URLs against the reference database to assess the initial credibility of the news source.
- Credibility scores of known news sources are periodically refreshed based on reputation data, historical accuracy, and crowd feedback.
- CheckMate periodically refreshes the reliability scores of previously analyzed news articles as new data and source assessments become available.
- CheckMate analyzes the factual accuracy of article content by performing reverse searches using external Google Custom Search API, to identify confirmatory or contradictory evidence.
- NLP models assess each article to detect biased or subjective language, flagging statements lacking credible references or containing conflicting information.

- Political articles undergo detailed sentiment and polarity analysis to classify them along a political spectrum (left-leaning, centrist, or right-leaning.
- CheckMate cross-references news articles against various sources, increasing the reliability score if corroborated and reducing it if significant discrepancies are identified.
- Each article receives a composite reliability score derived from:
    - Publisher credibility
    - Objectivity and subjectivity analysis of the news title
    - Objectivity and subjectivity analysis of the news article
    - Semantic similarity of the news article to various sources
    - Political bias analysis of the news article
    - Grammatical errors in the news article
- The system presents users with a breakdown of the score and a justification for each contributing factor.
- The system allows users to provide direct feedback regarding inaccuracies or biases detected in news articles, enhancing the overall effectiveness and accuracy of CheckMate through community-driven improvements.
- The system logs all administrative actions for transparency and auditability.
- CheckMate administrators have the capability to update news source reference databases and integrate new analysis data as it becomes available, ensuring ongoing accuracy and relevance.

2.2 Non-Functional Requirements

The system also meets the following non-functional requirements, ensuring overall quality and performance:

2.2.1 Usability

- The system provides an intuitive and user-friendly interface, allowing users to easily access each article's reliability score along with its rationale.
- The system displays clear visual indicators, including:
    - Color-coded reliability and credibility scores,

- Color-coded political bias indicators,
- Color-coded title and article objectivity scores
- Color-coded semantic similarity scores to indicate consistency with other sources,
- Color-coded spelling/grammar error rates to signal language quality.

● The system allows users to provide feedback on reliability of the articles and any possible bugs with minimal steps.

● The system is accessible on all major Chromium browsers (Google Chrome, Opera, Microsoft Edge and Brave) and other commonly used browsers such as Mozilla Firefox [5].

● The mobile application of CheckMate is compatible with both Android and iOS platforms optimized for responsive display across different screen sizes.

2.2.2 Reliability

● The system ensures accuracy and consistency in reliability scoring by continuously updating the reference database.

● User feedback is incorporated into the system to continuously improve the reliability score database and refine classification and scoring models over time.

2.2.3 Transparency

● The system provides clear justifications for reliability scores, detailing contributing factors such as subjectivity and political biases, contradictory evidence, and source credibility.

● The system allows users to review and contest assessments through a feedback mechanism, fostering accountability and continuous improvement.

2.2.4 Performance

● The system parallelizes the reverse search for 10 articles using the title of the main article as the query, executing these searches concurrently to improve efficiency and reduce latency.

● If the same article from the same news source has already been analyzed by another user, the system retrieves its cached reliability score and metadata directly from the database, minimizing redundant processing and improving user experience.

- Interactions with external APIs (for semantic similarity and reverse search) are optimized for low latency, ensuring fast and responsive system behavior even under high load.

2.2.5 Scalability

- Requests are distributed across multiple servers to prevent bottlenecks and ensure system responsiveness under concurrent load.
- Similar article searches and NLP tasks are executed in parallel, significantly reducing processing time and supporting higher throughput.
- The system is deployed on Google Cloud Run, which auto-scales containerized services based on incoming traffic.
- Each instance is provisioned with 4 vCPUs and 8 GB of memory, enabling efficient handling of resource-intensive NLP inference and API calls.
- Google Cloud Run automatically increases or decreases the number of instances in real-time, ensuring cost-effective scalability as user activity fluctuates.
- NLP tasks are processed asynchronously which ensures that time-intensive analyses do not hinder real-time user interactions.

2.2.6 Marketability

- The system is designed to appeal to a broad spectrum of users, including casual readers, journalists, and researchers, showcasing clear benefits such as quick reliability analysis and source credibility.
- The system's brand and messaging is crafted to highlight its role in combating misinformation, increasing public trust and adoption.
- The system offers flexible integration options freemium, premium and enterprise options to facilitate partnerships with news outlets, social media platforms, and research organizations.
- The system utilizes a highly reliable dataset comprising credible news sources, enabling it to compare the source of news articles against the reference database.
- The system has a mobile application compatible with both Android and iOS environments designed to provide a user-friendly and seamless experience on mobile devices.

2.2.7 Extendibility

● The system's architecture is modular, allowing new features to be integrated with minimal disruption.
● The codebase and database structures are organized in a manner that supports easy inclusion of updated or alternative fact-checking sources and models.

2.2.8 Security

● The system ensures that all collected user data, including user information, visited articles, and feedback, is handled in strict compliance with relevant data protection regulations (GDPR).
● The system employs end-to-end encryption for data transmission and securely stores user passwords using Werkzeug Security [6].

2.2.9 Maintainability

● The system's source code follows standardized coding conventions and best practices, making it easier for new developers to understand and modify.
● Comprehensive documentation will be provided, covering architecture, APIs, and deployment procedures.

2.2.10 Flexibility

● The system is implemented on all of the popular browsers (Google Chrome, Opera, Microsoft Edge, Brave, Mozilla Firefox) and mobile platforms (iOS, Android).
● Mobile application ensures users can seamlessly access fact-checking features and reliability scores on the go.

3. Final Architecture and Design Details

3.1 Overview

CheckMate is implemented as a cross-platform solution, designed to provide real-time news reliability assessments through both a browser extension and a mobile application. The browser extension is compatible with Chromium-based browsers such as Google Chrome, Opera, Microsoft Edge, Brave [5], and other common browsers such as Mozilla Firefox. Additionally, it is also available as a mobile app for both iOS and Android environments. The system architecture follows a modular and service-oriented design, where independent components handle distinct analytical tasks and communicate through RESTful APIs and share a centralized PostgreSQL database on Amazon RDS. CheckMate integrates multiple external services, core AI-based analyzers, a structured relational database, and scoring modules to automate misinformation detection.

The architecture is decomposed into:

- **Frontend Client Layer**: Browser extension (Chromium, Firefox) and mobile app (iOS/Android).
- **Backend Flask Services Layer**: Content scraping, article parsing, similarity analysis, and result aggregation.
- **Model Services Layer (Google Cloud Run)**: Stateless FastAPI [7] microservices hosting the bias classifier, objectivity classifier, and reliability scorer.
- **External APIs Layer**: Google Custom Search and Vertex AI Embeddings [8].

3.1.1 External Services

- **Amazon RDS** hosts the PostgreSQL database, which includes 5 tables: user information, news analyses results, news source reference tables, similar article logs and article search request logs. [9].
- **Google Cloud Run** servers provide the necessary endpoints for our backend and host NLP models which are political bias classification, objectivity classification and reliability models [10].
- **Google Vertex AI Embeddings API** returns the embeddings of the retrieved text articles to be later compared with cosine similarity to determine the respective similarity score.
- **Google Custom Search API** handles text-based reverse searches to validate article content [11].

3.1.2 Core Analyzers

The CheckMate architecture consists of 6 main components that work together to provide fact-checking and political bias classification:

- **Reliability Analyzer Component (Google Cloud Run):** NLP model that analyzes the reliability of the news using the similarity scores, credibility scores, political and objectivity classification results of the news article with grammatical error rate and title objectivity score. Predicts a final reliability score via a trained multilayer perceptron (MLP) model and returns it to the backend.
- **Reverse Search Component (Backend):** Reverse searches the title of the news article using Google Custom Search API to find 10 relevant news in parallel.
- **Similarity Analyzer Component (Backend):** Computes cosine similarity between the main article and reverse search results using Google Vertex AI text embeddings.
- **Language Analyzer (Google Cloud Run)**
    - Political Bias Classifier: Transformer-based model deployed via FastAPI on Cloud Run.
    - Objectivity Classifier: Custom RNN model returning objectivity probabilities for both article and title.
- **Source Analyzer (Backend):** Compares the news domain (normalized using tldextract) against the trusted_websites database [12] and returns a discrete credibility score.

3.1.3 Data & Databases

Amazon RDS handles the management of our database, which includes five tables, through PostgreSQL.

- trusted_websites table stores the known trusted news sources with their credibility labels gathered from [12].
- article_searches table holds the outcomes of each analysis with reliability scores, credibility scores, political bias classification results, similarity results, objectivity classification results and image analysis results with news URL and news title. It

also holds the last search date for the news' reliability score to get updated every week.

- users table holds the details of registered users, including attributes such as user ID, email address, password (hashed for security), creation date, subscription plan and Google ID (for sign in with Google option). This table is essential for managing user authentication, profile information, and activity tracking within the system.
- similar_articles table stores the 10 reverse-searched articles, including each article's title, URL, similarity score (from the similarity analyzer component), and search date. Reverse search is performed using the searched news text via the Google Custom Search API.
- article_requests table holds the news searched by the user with search date.

3.1.4 Score Calculation

The final reliability score displayed to the user is a result of aggregating several analytical outputs, each representing a specific aspect of the article's content quality, bias, and source trustworthiness. These individual scores are computed by independent components and NLP models and then synthesized using a central Reliability Score model. The weights assigned to each input component are empirically derived from a user perception survey conducted by our team, ensuring that the model reflects real-world expectations about what makes information trustworthy.

The input sub-scores that contribute to the overall reliability score are as follows:

- **Credibility Score:**
  This score is computed by the Source Analyzer Component, which references the news article's source against the *trusted_websites* table in our PostgreSQL database. The table contains curated metadata about domains, including their historical reliability, political inclination, and record of factual accuracy. The score is a discrete value ranging from 0-2 where 0 indicates uncredible, 1 indicates mixed and 2 indicates credible.

- **Similarity Score:**

  The Similarity Score Calculator uses Google Vertex AI text embeddings API to generate embeddings for the article's text. These are compared via cosine similarity to embeddings of 10 related articles retrieved using Google Custom Search API. The average similarity score reflects how closely the input article aligns with information published by other reputable sources. The score is a continuous value ranging from 0-1 where 1 indicates the highest similarity.
- **Political Bias Score:**

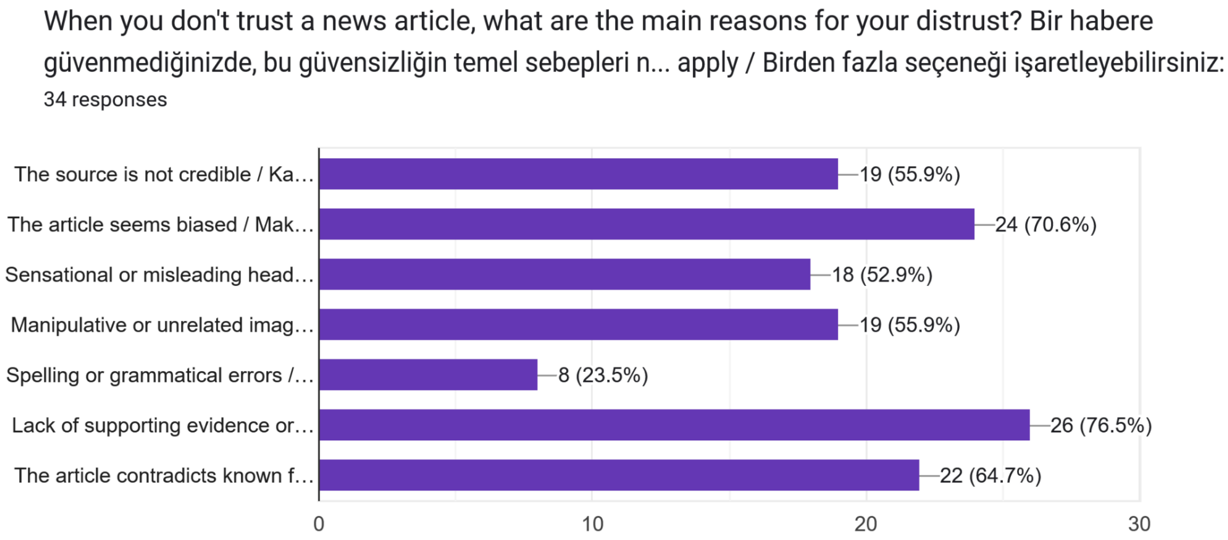
  This score is the result of political bias classification model that evaluates whether the article leans left, right, or remains neutral indicated by central leaning. The model considers phrasing, lexical patterns, and ideological indicators. Political slant affects the reliability score depending on its extremity. The score is a continuous value ranging from 0-1 where 0 is designated as far-left and 1 is designated as far-right and 0.5 is set as centrist-view.
- **Objectivity Score:**

  This score quantifies how objective or subjective the article's language is. Articles that rely on emotional, vague, or manipulative language are marked as more subjective and penalized accordingly in the final calculation. The score is a continuous value ranging from 0-1 where 1 indicates the highest objectivity.
- **Title Objectivity Score:**

  This score evaluates whether the article's title is written in a neutral tone or if it contains clickbait, emotionally charged language, or exaggerated claims. Since headlines heavily influence reader perception, biased or manipulative titles result in a reduced reliability score. Same scoring method is used as the regular objectivity score.
- **Grammatical Error Rate:**

  This functionality detects and quantifies grammatical errors throughout the article. Poor grammar often signals automated or low-quality authorship and is treated as a red flag for misinformation. It adds a valuable layer of quality assurance.

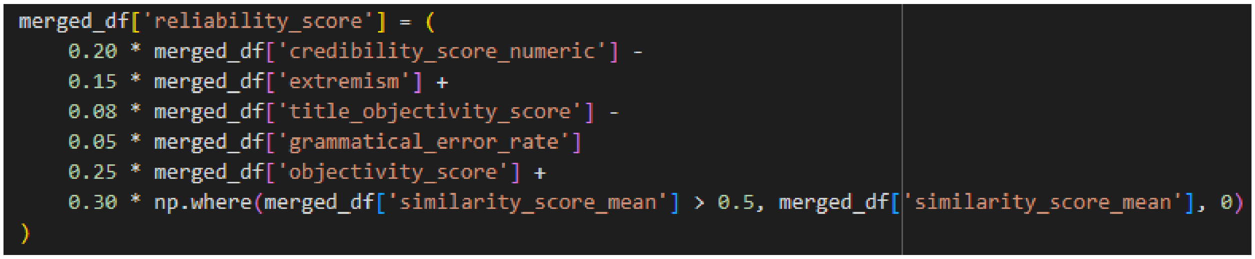
All the scores mentioned above are aggregated by the Reliability Score Calculator, which applies weighted factors informed by our survey data to compute a final normalized score between 0 and 1 where 1 is the highest reliability. This value is later converted to a percentage for user-friendliness.



When you don't trust a news article, what are the main reasons for your distrust? Bir habere güvenmediğinizde, bu güvensizliğin temel sebepleri n... apply / Birden fazla seçeneği işaretleyebilirsiniz:

34 responses

Fig. 1: User perception survey conducted by our team [13]



```
merged_df['reliability_score'] = (
    0.20 * merged_df['credibility_score_numeric'] -
    0.15 * merged_df['extremism'] +
    0.08 * merged_df['title_objectivity_score'] -
    0.05 * merged_df['grammatical_error_rate']
    0.25 * merged_df['objectivity_score'] +
    0.30 * np.where(merged_df['similarity_score_mean'] > 0.5, merged_df['similarity_score_mean'], 0)
)
```

Fig. 2: Weights assigned to each input component in Reliability Score Model are empirically derived from our user perception survey conducted by our team

Once computed, the reliability score shall be presented to the user with a color-coded indicator:

- Green for reliable articles
- Yellow for mixed or moderately reliable articles
- Red for unreliable articles

All features are passed to the Reliability Scoring Service on Cloud Run, which returns a value ∈ [0,1] interpreted with:

- Green: Reliable (≥ 0.75)
- Yellow: Mixed (0.45 – 0.74)
- Red: Unreliable (< 0.45)

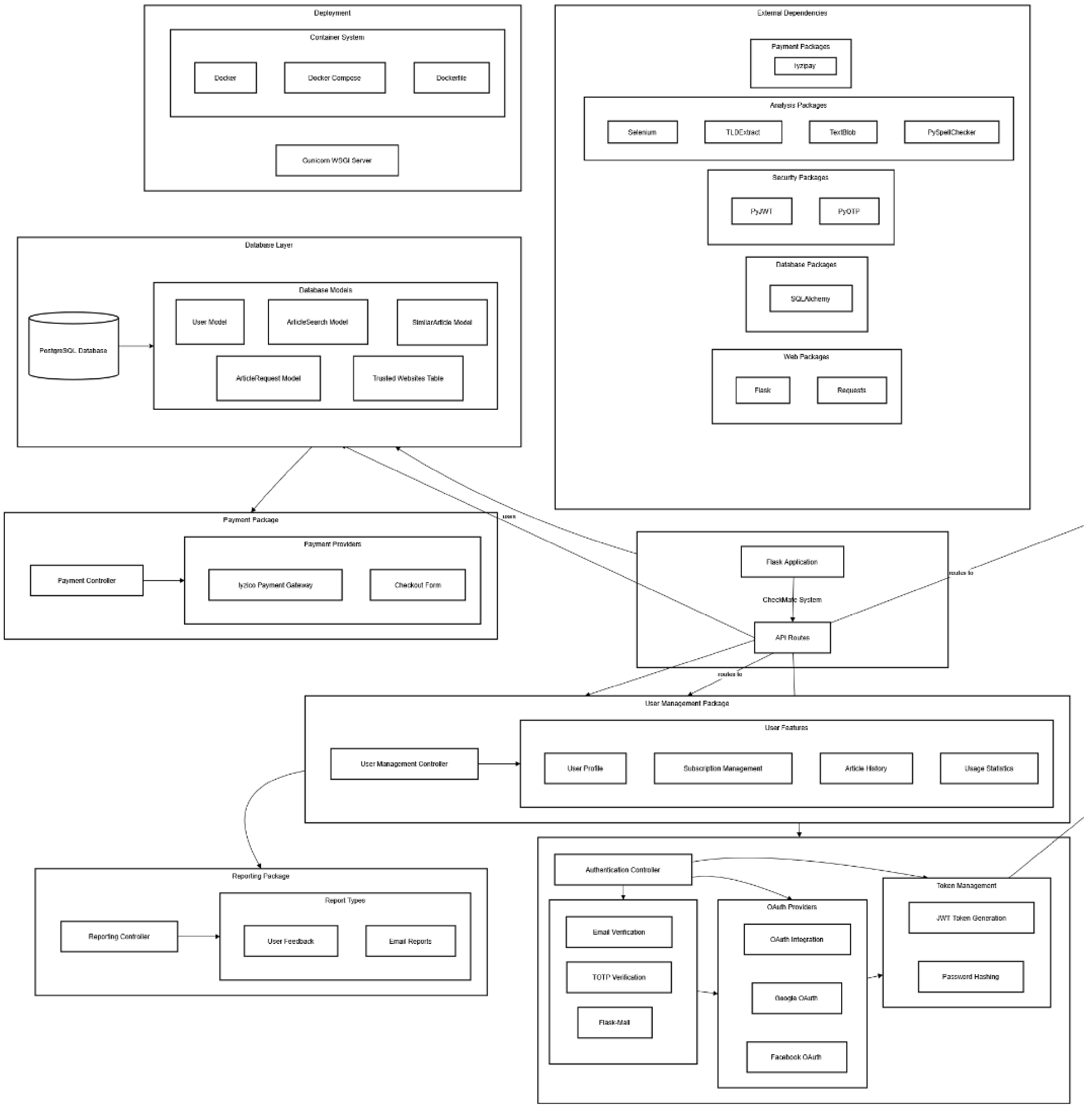
A score rationale breakdown is shown to the user, along with interpretations of objectivity, bias, and trust indicators. This structured, explainable scoring pipeline allows CheckMate to provide fast, meaningful, and evidence-based evaluations of online news articles at scale.

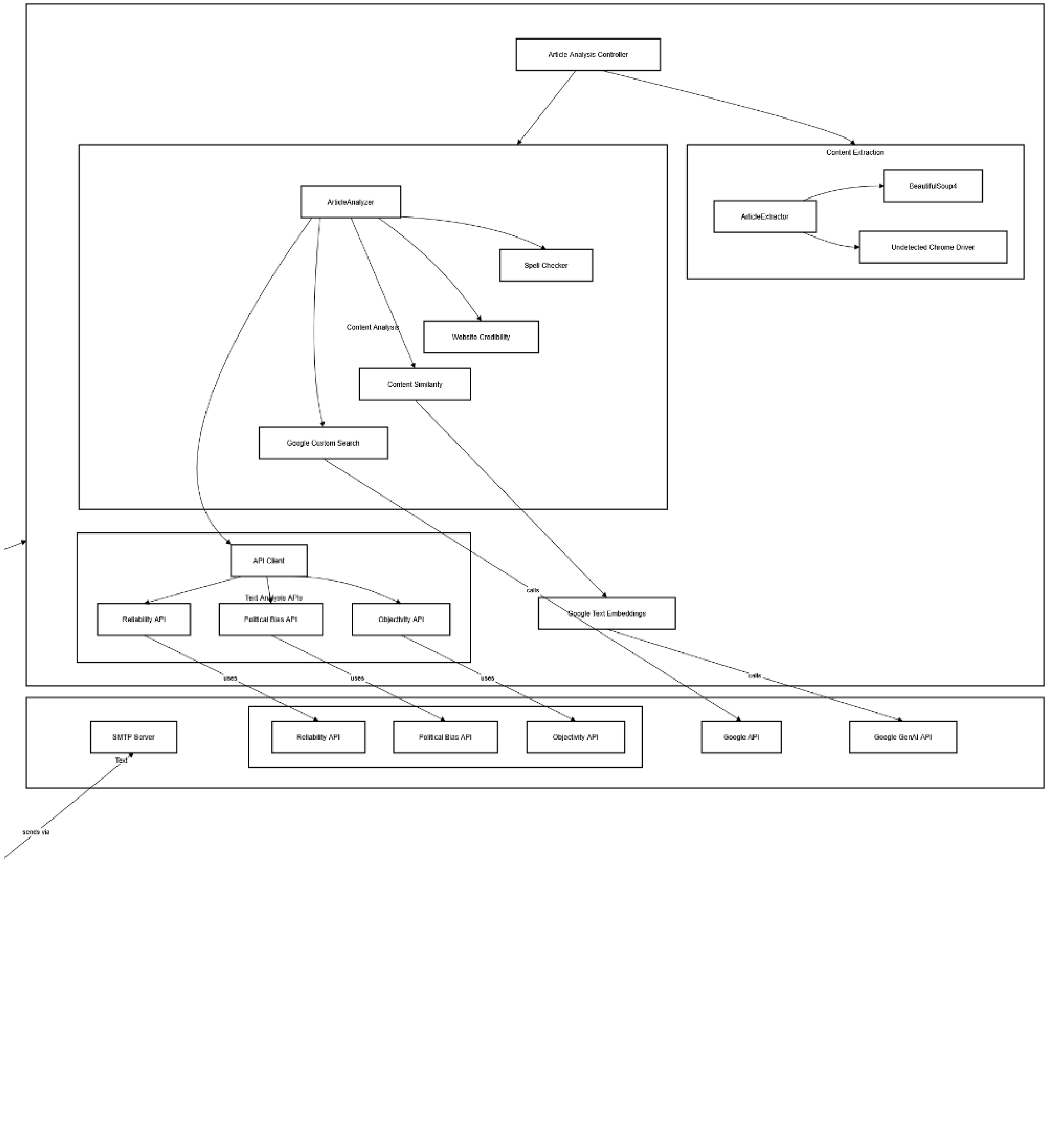
3.2 Subsystem Decomposition

CheckMate's architecture is decomposed into:

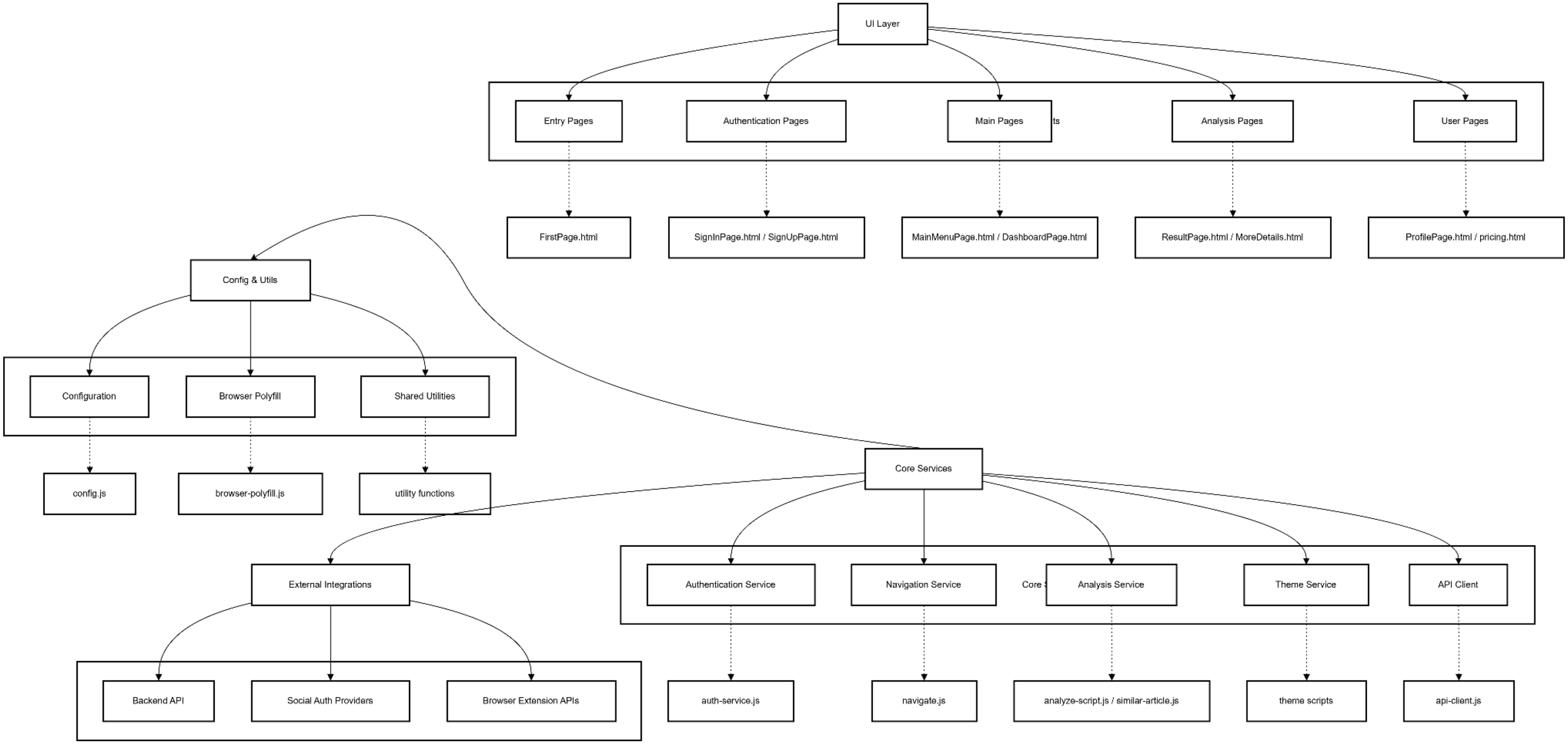
- **Frontend Clients Layer**: Browser extension (Chromium, Firefox) and mobile app (iOS/Android).
- **Backend Flask Services Layer**: Content scraping, article parsing, similarity analysis, and result aggregation.
- **Model Services Layer (Google Cloud Run)**: Stateless FastAPI microservices hosting the bias classifier, objectivity classifier, and reliability scorer.
- **External APIs Layer**: Google Custom Search and Google Vertex AI text embeddings.

Backend System Decomposition Diagram

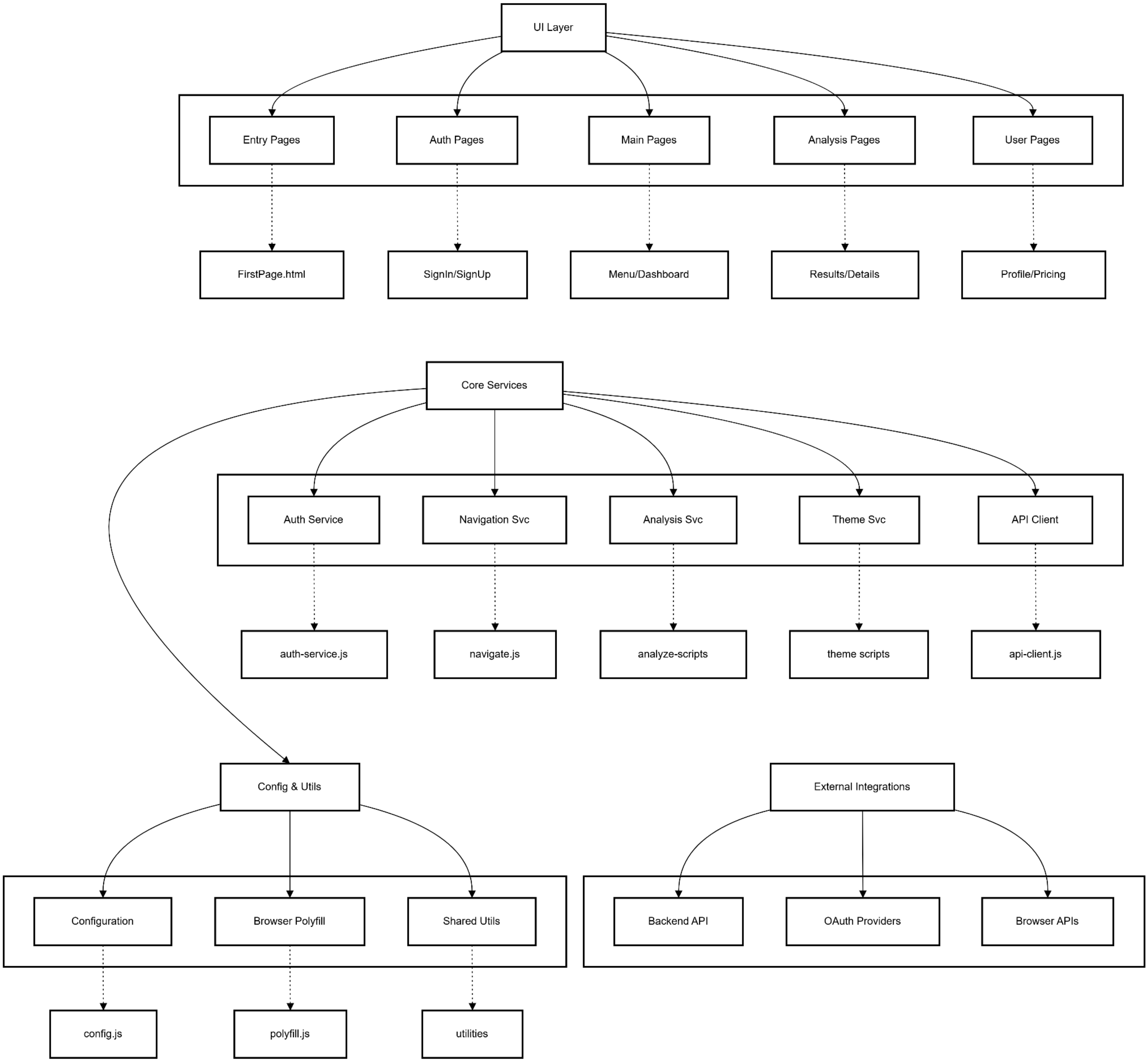



Article Analysis Controller
Content Extraction
BeautifulSoup4
ArticleExtractor
Undetected Chrome Driver
ArticleAnalyzer
Spell Checker
Content Analysis
Website Credibility
Content Similarity
Google Custom Search
API Client
Text Analysis APIs
Reliability API
Political Bias API
Objectivity API
calls
Google Text Embeddings
uses
uses
uses
calls
SMTP Server
Reliability API
Political Bias API
Objectivity API
Google API
Google GenAI API
Text
sends via

Browser Extension Frontend System Decomposition Diagram



Mobile App Frontend System Decomposition Diagram



3.2.1 Frontend Client Layer (Google Cloud Run)

This layer provides the user-facing interfaces for CheckMate:

- **CheckMate Browser Extension**: Compatible with Chromium-based browsers (Chrome, Edge, Opera, Brave) and Mozilla Firefox.
- **CheckMate Mobile Application**: Available for both iOS and Android platforms.

Each client includes modules for:

- Authentication (sign-in/sign-up)
- Dashboard access
- Search history visualization
- Detailed article analysis results
- User profile and subscription management

All user actions are transmitted as HTTPS requests to a shared backend, ensuring consistent experience across platforms.

3.2.2 Backend Flask Layer (Google Cloud Run)

The core backend logic is implemented using Flask and deployed as a **Google Cloud Run service**, acting as the bridge between the frontend and the deeper processing layers. It includes the following subsystems:

- **Backend Server**: Manages API routing, session validation, and communication with databases and model services.
- **Article Extractor**: Parses article text, title, metadata, and images using hybrid scraping and fallback browser automation.
- **Reverse Search Engine**: Performs semantic retrieval via Google Custom Search API, fetches articles, and filters domains.
- **Similarity Analyzer**: Computes cosine similarity between articles using embeddings from the Vertex AI API.

- **Website Credibility Checker**: Cross-references domains against the trusted source database to assign a credibility label.

This layer is shared across all frontend clients and interacts with a centralized PostgreSQL database on Amazon RDS, maintaining user search logs, results, and system metadata.

3.2.3 Model Services Layer (Google Cloud Run)

Stateless FastAPI microservices, independently deployed on Google Cloud Run, handle ML model inference for:

- **Political Bias Classification Model:** Uses a transformer-based classifier to determine left/center/right orientation.
- **Objectivity Classification Model:** Evaluates linguistic neutrality of both article text and title.
- **Reliability Scoring Model:** Aggregates features (bias, objectivity, credibility, similarity, grammar) to produce a final reliability score using a trained MLP model.

These services are modular and independently scalable, enabling efficient parallel processing and isolated updates.

3.2.4 External APIs Layer (Google Cloud Run)

The External APIs layer enhances CheckMate's capabilities by integrating third-party services:

- **Google Custom Search API:** Retrieves web results related to fact-checking queries [11].
- **Google Cloud Vision API:** Facilitates image recognition and analysis for reverse searching [14].
- **Google Vertex AI Text Embeddings API:** Retrieves the embeddings for the searched article and the articles returned by the Reverse Search Component.

3.2.5 Interactions Between Layers

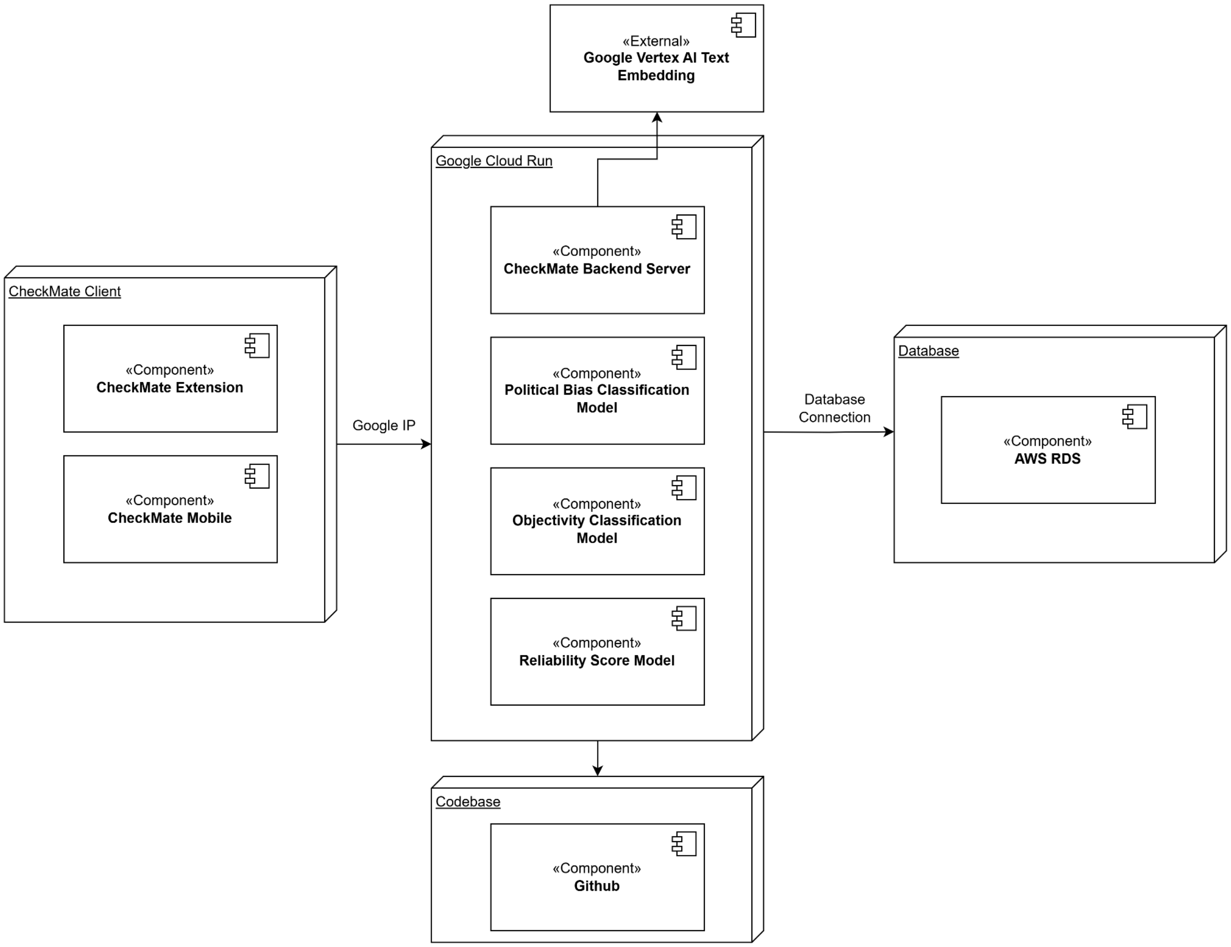
Users interact with the system through the Client Layer, which includes the CheckMate browser extension and mobile application. These interfaces capture user actions such as article submission, feedback, and profile management and send them as HTTP requests to the backend via secure API calls.

The Backend Logic Layer, hosted on Google Cloud Run, receives and processes these requests. It coordinates the necessary operations by invoking relevant modules, including the article extractor, reverse search engine, and AI-powered analyzers. These services evaluate the credibility, similarity, political bias, objectivity, and grammatical quality of the article.

All processed data including user input, analysis results, metadata, and feedback is stored in a PostgreSQL relational database hosted on Google Cloud. This data is used for tracking history, improving model performance, and ensuring consistent scoring across future user interactions.

This modular, multi-layered architecture enables CheckMate to deliver accurate and real-time misinformation detection, while supporting scalability, performance optimization, and easy system maintenance.

3.3 Hardware/Software Mapping



The CheckMate system consists of four main modules in its hardware/software mapping: CheckMate Client, Google Cloud Run, RDS, and the Codebase. The CheckMate Client module includes the CheckMate Extension and CheckMate Mobile, which are for client interaction. These clients communicate with the CheckMate Backend Server, which is deployed on Google Cloud Run. The backend is responsible for handling requests and running machine learning models: Objectivity Classification Model, Political Bias Classification Model, and Reliability Score Calculator Model to analyze and classify input data. The AWS RDS module serves as the database, storing user interactions, analysis results, and system data. Additionally, the Codebase module is hosted on GitHub, where the source code is maintained and updated. This structure ensures that CheckMate is scalable, secure, and capable of handling multiple user requests efficiently.Given the modular architecture of CheckMate, different system components are deployed across cloud platforms and client environments optimized for their respective workloads.

Backend (Flask Services)

- **Hosting Platform**: Google Cloud Run
- **Deployment Model**: Containerized via Docker
- **Operating System**: Linux (inside container runtime)
- **Scaling**: Automatically scales up to 10 instances based on concurrent requests
- **Resources per Instance**:
  - Memory: 8 GB RAM
  - CPU: 4 vCPU

Model Services (FastAPI Microservices)

- **Hosting Platform**: Google Cloud Run
- **Services Hosted**: Political Bias Classifier, Objectivity Classifier, Reliability Scorer
- **Deployment Model**: Stateless containerized microservices
- **Resources per Instance**:
  - Memory: 8 GB RAM
  - CPU: 4 vCPU
- **Scaling**: Independent autoscaling per service up to 5 concurrent instances

Frontend (Mobile App & Web Extension)

- **Mobile App Platform**: Capacitor-based hybrid app for iOS and Android
- **Hosting (Web Extension)**: Client-side execution within user's browser
- **Compatibility**: Supports Chromium-based browsers (Chrome, Edge, Brave) and Firefox
- **Execution Environment**: Local device or browser runtime
- **Resource Utilization**: Lightweight; depends on the end-user device but optimized for low memory and CPU usage

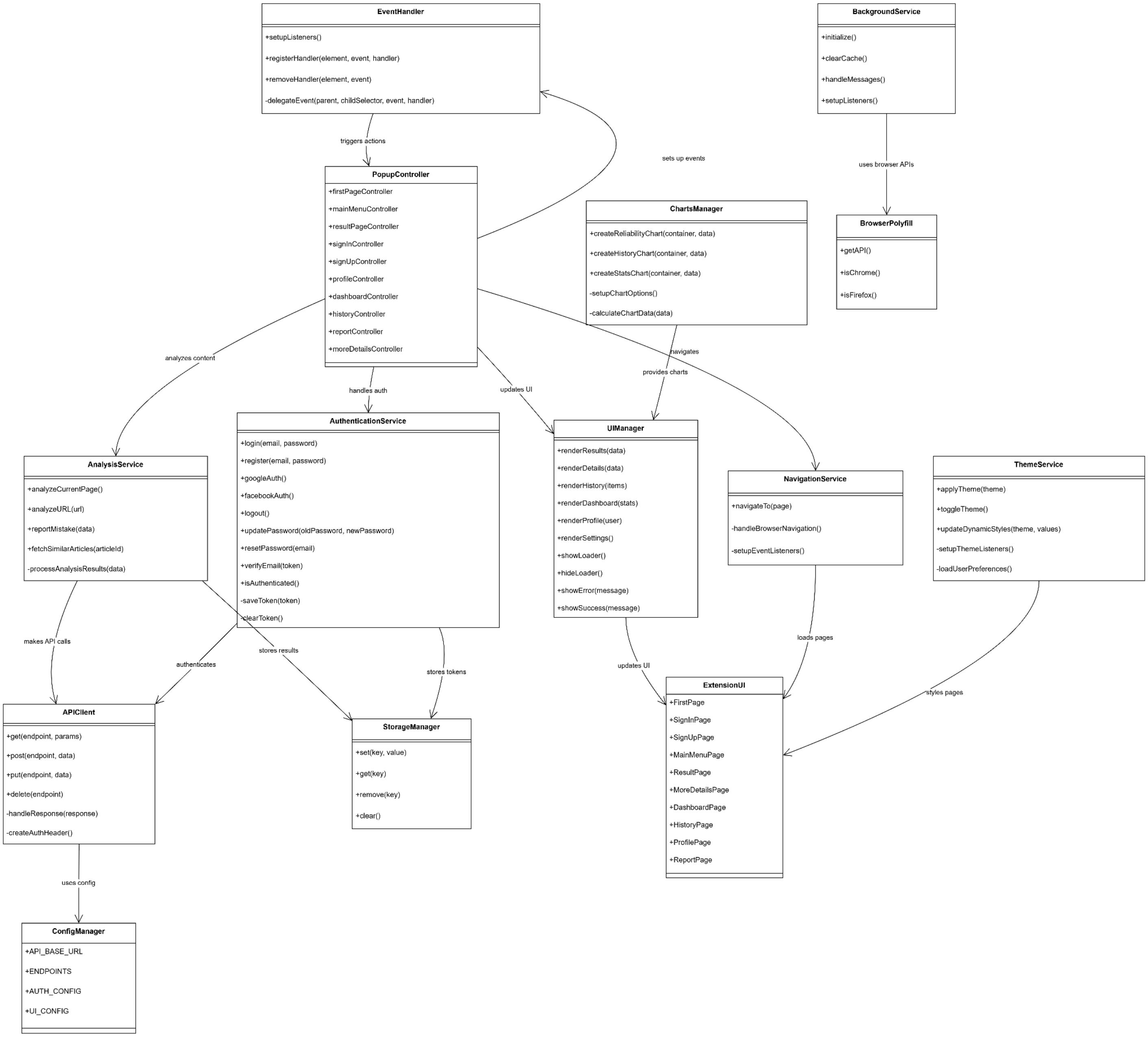
Temporary Hardware for Model Training

- **Platform**: AWS EC2
- **Instance Type**: EC2 G5
- **Resources**:
  - GPU: 1× NVIDIA A10
  - CPU: 8 vCPUs
  - RAM: 64 GB

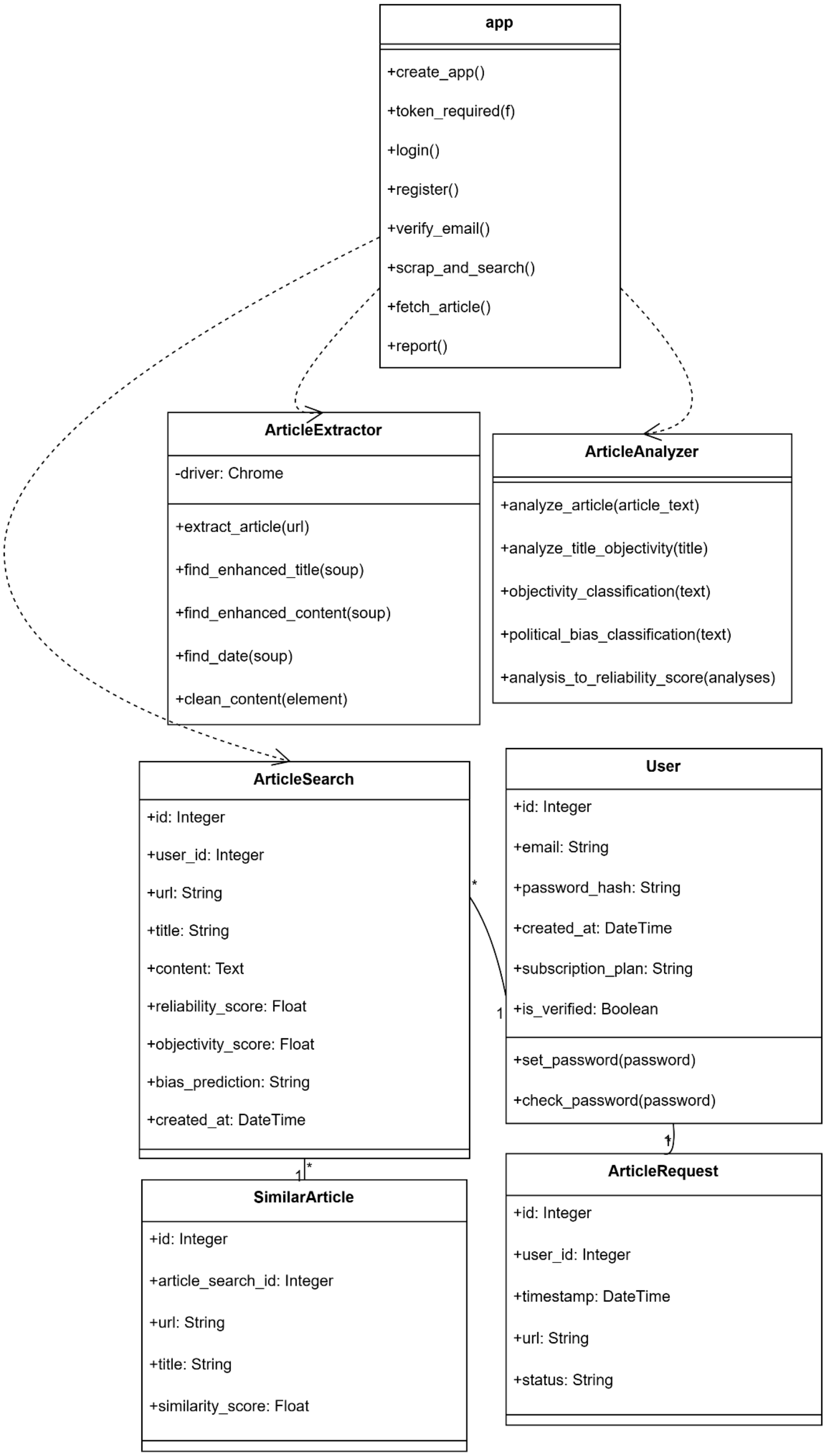
All components are deployed on scalable cloud infrastructure, allowing resource limits to be increased as needed. The estimated operational cost is approximately $1 per user per month over an 8 month period, covering inference, storage, and scaling overhead.

3.4 Class Diagrams

3.4.1 Browser Extension Frontend Class Diagram



3.4.2 Backend Class Diagram



3.5 Access Control and Security

CheckMate implements a tiered access control system based on Free, Premium, and Enterprise subscription levels. Security is enforced through a combination of authentication, authorization, encrypted communication, and secure storage mechanisms:

- **Authentication & Tokenization**:
    - Users authenticate via email/password, Google OAuth, or Facebook login.
    - Upon successful login, a JWT token is issued and attached to subsequent requests for session continuity.
    - Tokens are signed and time-limited, with expiration adjusted based on "remember me for 30 days" preference.
- **Role-based Access Control (RBAC)**:
    - Sensitive operations (subscription plan updates, payment processing, administrative actions) are restricted to verified roles.
    - Each incoming request is authorized against the user's plan and verification status using custom backend decorators (@token_required).
- **Secure Storage**:
    - All user data (email, hashed passwords, OAuth IDs, subscription plans, usage history) is stored in Amazon RDS PostgreSQL, protected by VPC, IAM-based connection restrictions, and parameterized SQL queries.
    - Passwords are hashed using bcrypt-based hashing via werkzeug.security.
- **Request Limits & Rate Control**:
    - API usage caps are enforced per tier via runtime request counting (DAILY_USAGE logic).
- **Security Practices**:
    - HTTPS is enforced across all endpoints.
    - Email verification via TOTP (time-based one-time password) is implemented for account integrity.
    - Payment handling is delegated to İyzico, a PCI-DSS-compliant provider, using tokenized card data.

3.6 Subscription Management Module

The Subscription Management module governs access to system features based on the user's subscription tier:

- **Subscription Plan Limits**:
  - **Free**:
    - Up to 20 analyses/day
    - Limited output: summarized scores only
  - **Premium**:
    - Up to 100 analyses/day
    - Full access to breakdowns: similarity details, objectivity score details of title and article, political bias, grammar errors
  - **Enterprise**:
    - Unlimited access
    - Access to private bulk-analysis API and enhanced dashboards
- **Middleware**:
  - On every API call (/scrap_and_search, /user/stats), the user's plan is checked from the users table.
  - Daily usage is calculated by querying article_requests joined with timestamps.
  - Throttling is enforced using app.config['DAILY_USAGE'] and custom exception handlers.
- **Payment & Subscription Plan Downgrade Handling**:
  - Payments are handled securely via İyzico API with card tokenization (cardToken, cardUserKey) and secure HMAC-based authentication.
  - Subscription upgrades and downgrades are reflected in real-time by updating the subscription_plan field in the database.
  - On payment failure or cancellation:
    - Users are automatically reverted to Free plan.
    - Their access is restricted immediately without deleting historical data.
- **Administrative Control**:
  - Admins can manually assign plans via a secure endpoint (not exposed to users).

  - Enterprise onboarding can be customized through email invitations or private contract uploads.

4. Development/Implementation Details

4.1 Frontend

The frontend of CheckMate is implemented separately for the browser extension and the mobile app, both communicating with the shared backend via secure RESTful APIs.

4.1.1 Application Shell & Navigation

All pages (Main Menu, Analyze, Results, History, Profile, Dashboard, Email Verification, Sign In/Up, Report) share:

- A common navigateTo(page) helper for intra-extension navigation.
- localStorage to persist the JWT token, user email, plan, and the most recent analysis results.
- Conditional redirects based on authentication state.

4.1.2 Authentication & Signup

- Sign-In (signin-script.js): Captures email, password, and "Remember Me," shows inline validation/errors, swaps the button for a spinner during the POST /user/login call, stores token, userEmail, and userPlan on success, and navigates to MainMenuPage.html. Supports Google OAuth (authService.googleSignIn()) and Facebook OAuth via chrome.identity.launchWebAuthFlow .
- Sign-Up (signupscript.js): Toggles password visibility (eye/eye-slash icons), validates password vs. confirm-password, then calls authService.register(email, name, password) (using email as name), and on success redirects to SignInPage.html .

4.1.3 Email Verification & Password Reset

- Verification Flow (email-verification.js & script.js):

  - Email Entry: User inputs email; clicking “Send Code” POSTs to /user/send-verification-code, shows success/error messages, hides email form, and reveals code-entry section.
  - Code Confirmation: Validates a six-digit code, POSTs to /user/verify-email, then on success shows a notification and redirects to SignInPage.html.
  - UI Logic: Toggling between sections, input validation (email regex, numeric code), and user feedback via text messages and CSS classes .

4.1.4 Article Analysis & Results

- Analysis Trigger (analyze-script.js): Queries the active tab’s URL, allows manual entry, then POSTs { url } to /scrap_and_search with the JWT header. Stores the JSON response in localStorage.analysisResults and redirects to ResultPage.html .
- Result Rendering (similar-article.js): Reads analysisResults and populates:
  - Reliability, Objectivity, Title Objectivity, Bias scores (percentages, color classes).
  - Website Credibility label.
  - Similar Articles list with titles, links, and similarity badges.
  - Image Analysis entity lists and thumbnails.
- Result Placeholders (result-scripts.js): “More Details” and “Report Mistake” buttons currently fire alerts, marking spots for future UI enhancements .

4.1.5 Reporting Mistakes (report.js)

Handles user feedback on analysis errors:

- Toggles custom radio button styles (active class).
- Pre-selects “Reliability Score Issue” if reportType was set in localStorage.
- On form submit, gathers reportType and message, POSTs to /report with JWT, shows a loading spinner, and displays success/failure notifications.
- Resets form and radio styling on success .

4.1.6 Search History (history-script.js)

- Fetches GET /user/searches with JWT, reverses to show newest first.

- Formats timestamps, renders clickable cards (title, URL, date, reliability badge).
- On click, disables other cards, shows a spinner, fetches /article/{id}, reloads that analysis into localStorage, and navigates to ResultPage.html .

4.1.7 User Profile & Subscription

- Profile Display (profile-script.js): Reads userEmail and userPlan from localStorage, updates the DOM with plan badges (“Free”, “Premium”, “Enterprise”), gradients, and hover animations; wires “Change Password” and “Upgrade”/“Manage Plan” buttons .
- Payment Logic (payment-service.js): Exports PaymentService with methods to initialize (/cf/initialize), query (/cf/query), fetch current plan (/user/stats), and update the plan (/user/update-plan), all attaching the JWT to requests .

4.1.8 Dashboard (dashboard.js)

Provides a real‑time usage and accuracy dashboard:

- Initialization: Fetches GET /user/stats, normalizes percentage values, or falls back to mock data on error.
- UI Updates:
  - Progress Bar: Animates daily usage vs. limit, color‑codes based on thresholds.
  - Stat Cards: Daily limit, articles analyzed today, total articles, and Reliability over selectable ranges (week/month/year).
  - Animations: Fade‑in and count animations for card values.
- Charting: Uses Chart.js to render a bar chart of articles‑analyzed distribution for the selected time range, with dynamic data processing for weekly, monthly, and yearly views.
- Interactivity: Time‑range selector, menu/profile/history buttons, and auto‑refresh every five minutes .

Together, these scripts deliver a seamless, authenticated Chrome-extension UI for CheckMate

4.2 Backend

The backend is built using Flask and deployed on Google Cloud Run, handling core processing, external API integration, database interaction, and orchestration of model services.

4.2.1 Dependencies & Configuration

All third-party libraries are declared in requirements.txt, including Flask and its extensions (Flask-CORS, Flask-Mail, Flask-SQLAlchemy), environment management (python-dotenv), authentication (PyJWT, pyotp, itsdangerous), headless browsing (undetected-chromedriver), HTML parsing (beautifulsoup4), database drivers (psycopg2-binary), and Google API clients (google-auth, google-genai) .

Configuration values- database credentials, JWT secret, Google API keys, mail server details, and İyzico payment secrets- are all loaded via load_dotenv() in both app.py and init_db.py, ensuring no sensitive data is hard-coded .

4.2.2 Data Models & Database Initialization

In models.py, we define four core tables with SQLAlchemy:

- **User**: stores credentials, subscription plan, social IDs, verification status, and relations to requests.
- **ArticleSearch**: logs each analyzed URL, title, and its scores (reliability, credibility, objectivity, bias) alongside a timestamp.
- **SimilarArticle**: holds titles, URLs, and similarity scores for related articles.
- **ArticleRequest**: a join table tracking which users requested which analyses .

The script init_db.py uses these model definitions to create all tables in PostgreSQL by spinning up a minimal Flask app and calling db.create_all() within its application context .

4.2.3 Article Extraction & Website Credibility

Raw content is fetched and cleaned by the ArticleExtractor class in ArticleExtractor.py. It launches an undetected Chrome instance, handles retries and cookie management, scrolls dynamically, and uses BeautifulSoup with enhanced and fallback selectors to pull out titles, paragraphs, dates, and images .

Before any NLP analysis, website_checker.py looks up the article's domain against a trusted_websites table to assign a preliminary credibility score via a simple SQL query .

4.2.4 Application Factory & Extensions

Most of the backend service is in app.py. The create_app() function wires up:

- **Flask** with environment-driven settings.
- **Flask-CORS** to allow our Chrome extension and other clients to hit the API.
- **Flask-Mail** for all email flows.
- **SQLAlchemy** bound to a PostgreSQL URI built from env vars.

All of these initializations happen before any routes are registered .

4.2.5 Authentication & Authorization

A token_required decorator extracts a Bearer token from the Authorization header, decodes it with current_app.config['JWT_SECRET_KEY'], and fetches the corresponding User. Invalid, missing, or expired tokens immediately yield HTTP 401 responses .

For email-based verification, we integrate itsdangerous.URLSafeTimedSerializer (link tokens) and a TOTP-based flow (pyotp wrapped by TOTPVerification) to send and confirm one-time codes via email.

4.2.6 Core Fact-Checking Endpoint: /scrap_and_search

This POST route orchestrates the complete analysis pipeline:

- **Validate input** for a JSON body containing "url".
- **Fetch website credibility** via check_website_score.
- **Cache lookup**: if the URL was seen before, return existing ArticleSearch and SimilarArticle records, logging a new ArticleRequest if needed .

- **Fresh extraction**: instantiate ArticleAnalyzer (wrapping the extractor and NLP logic), validate the scraped content, and call its get_similar() method to retrieve semantically similar articles via Google Custom Search + Vertex AI Text Embeddings .
- **Persist results**: create new rows in ArticleSearch, ArticleRequest, and bulk-insert all SimilarArticle entries within one DB transaction for atomicity .
- **Return JSON** with reliability, objectivity, bias scores, website credibility, and an array of similar articles.

Detailed debug print() statements pepper the flow to aid troubleshooting in container logs.

4.2.7 Supplementary Endpoints

Beyond analysis, app.py implements:

- **User lifecycle**: registration, login (with JWT issuance), password updates, forgotten-password resets, and social OAuth (Google/Facebook) flows.
- **Subscription & payments**: İyzico integration via HMAC-signed requests (generate_iyzico_v2_headers), endpoints to initialize (/cf/initialize) and query (/cf/query) checkout forms, and upgrade plans.
- **Analytics & history**: /user/stats aggregates counts and average accuracy over various timeframes; /user/searches and /article/<id> let users retrieve past analyses.
- **Reporting**: a simple /report endpoint for bug and feedback submissions.

All routes leverage the same application context, shared DB session, and authentication mechanisms for a cohesive, maintainable codebase.

4.3 Models

The core NLP models in CheckMate perform political bias classification, objectivity scoring, and reliability scoring, each accessed as independent microservices.

4.3.1 Technology Stack and Deployment Environment

The backend responsible for handling model inference in CheckMate is developed using:

- Language: Python 3.10
- Framework: FastAPI for asynchronous REST API handling
- ML Libraries:
  - *transformers* (HuggingFace) for the political bias classifier
  - *joblib* for loading serialized scikit-learn models
  - *nltk* for tokenization in the subjectivity classifier
  - *torch* (PyTorch) for deep learning model inference
  - *numpy* for numerical computation
- Runtime Optimization: *asyncio* with *ThreadPoolExecutor* is used for concurrent execution to maintain responsiveness while running CPU-bound inference tasks.
- A Dockerfile for deployment.

4.3.2 API Exposure

The *checkmate-api* service exposes the following HTTP endpoints:

- GET /: Health check route
- POST /subjectivity: Runs sentence-level subjectivity analysis
- POST /political: Classifies political bias as left, center, or right
- POST /reliability: Returns a final reliability score for a news article

4.3.3 Subjectivity Classifier

The subjectivity module uses a custom neural classifier for sentence classification.

- **Module**: SubjectivityClassifier
- **Model**: TensorFlow.tf model trained on subjective vs. objective sentences
- **Embeddings**: Pre-trained GloVe embeddings [15] (glove.6B.50d.word2vec.txt)
- **Processing Pipeline**:
  - The input article is split into sentences using nltk.tokenize.sent_tokenize.
  - Each sentence is converted into an embedding.
  - The model returns:
    - *subjective_sentences*: list of subjective sentences

- *objective_sentences*: list of objective sentences
- *subjectivity_prob*: average probability of subjectivity across the article
- *class_label*: majority class (subjective or objective)

- **Accuracy**: Approx. **90.5%** on balanced validation data

Epoch	**Precision (%)**	**Recall (%)**	**F1 Score (%)**	**Accuracy (%)**
29	91.48	88.61	90.02	90.09
30	90.57	90.57	90.57	90.49
31	85.44	94.50	89.74	89.10
32	90.43	90.96	90.70	90.58
33	90.57	90.57	90.57	90.49
34	89.58	91.16	90.36	90.19
35	90.00	91.94	90.96	90.78
36	90.14	91.55	90.84	90.68
37	90.69	89.98	90.34	90.29
38	91.88	86.64	89.18	89.40
39	87.89	94.11	90.89	90.49

Execution is delegated to a *ThreadPoolExecutor* to avoid blocking the main FastAPI event loop.

4.3.4 Political Bias Classifier

This module classifies articles into Left, Center, or Right political orientation.

- **Model Architecture**: BERT (base-uncased) fine-tuned for 3-class classification
- **Tokenizer**: HuggingFace AutoTokenizer
- **Inference Method**:
    - Input text is tokenized with truncation and padding (max_length=512).
    - The model outputs logits which are softmax-normalized into probability distributions.

    - The predicted label is the argmax of the logits.
- **Labels**: 0 → Left, 1 → Center, 2 → Right
- **Accuracy**: Approx. 86% on balanced validation data
- **Output**: JSON response containing:
    - prediction: predicted class (Left, Center, Right)
    - probabilities: softmax-normalized confidence scores for each class

This classification is used as a core feature in the reliability calculation model.

4.3.5 Reliability Score Model

The reliability score is calculated based on bias, objectivity, source credibility, and semantic similarity scores. The /reliability endpoint accepts a POST request with the following JSON payload:

```
{
  "bias_probs": {"Left": 0.2, "Center": 0.6, "Right": 0.2},
  "objectivity_score": 0.82,
  "credibility_score": "credible",
  "similarity_scores": [0.73, 0.81, 0.67, ...]
}
```

4.3.5.1 Preprocessing and Feature Engineering:

- **Extremism**: |Left - Right| – captures ideological polarity
- **Credibility Score** is mapped to: "credible" → 1.0, "mixed" → 0.5, "uncredible" → 0.0
- **Similarity Scores** (10 scores from reverse search results):
    - Mean is used only if above a 0.5 threshold (ensuring article is sufficiently verifiable).
    - sim_mean, sim_max, sim_min, and sim_std are extracted.

The input features are a 10-dimensional vector:

```
[center_prob, left_prob, right_prob, extremism, objectivity_score, credibility_score,
 sim_mean, sim_max, sim_min, sim_std]
```

4.3.5.2 Model:

- **Algorithm**: Multi-Layer Perceptron (MLP) trained via *scikit-learn*
- **Scaler**: MinMaxScaler applied to normalize input features
- **File Formats**:
    - Model: mlp_model.joblib
    - Scaler: scaler.joblib
- **Output**:
    - A float score in the range [0, 1] representing final reliability
    - Rounded to 4 decimal places
    - Internally clamped to stay within valid bounds
    - Later visualized as a percentage in the UI (e.g., 0.91 → 91%)

5. Test Cases and Results

Test ID	TC_1.1 – Add Extension	Category	Functional ▾	Severity	Critical ▾
Objective	Verify the extension can be installed successfully.				
Steps	User installs the Checkmate extension.				
Expected	Extension installation succeeds with no errors (the extension icon is visible).				
Result	Pass				
Date	01.03.2025				

Test ID	TC_1.2 – Sign Up with Valid Data	Category	Functional ▾	Severity	Critical ▾
Objective	Verify that a new account is created with valid data.				

Steps	User clicks Sign Up. User enters valid name, email, phone number, password, confirm password. User clicks Sign Up.
Expected	The user is added to the database otherwise an appropriate error is given.
Result	Pass
Date	01.03.2025

Test ID	TC_1.3 – Sign Up with Missing Fields	Category	Functional ▾	Severity	Major ▾
Objective	Ensure an error is shown if any required fields are omitted.				
Steps	User clicks Sign Up. User leaves at least one required field blank (e.g., phone or password). User clicks Sign Up.				
Expected	Error message appears prompting the user to fill the missing field(s).				
Result	Pass				
Date	01.03.2025				

Test ID	TC_1.4 – Sign Up with Invalid Email	Category	Functional ▾	Severity	Major ▾
Objective	Confirm that an invalid email format is not accepted.				

Steps	User clicks Sign Up. User enters an invalid email (e.g., “test@” or missing “@”). User fills other fields correctly. User clicks Sign Up.
Expected	An error message is displayed (e.g., “Invalid email format”); account is not created.
Result	Pass
Date	01.03.2025

Test ID	TC_1.5 – Sign Up with Password Mismatch	Category	Functional ▾	Severity	Minor ▾
Objective	Ensure password and confirm password must match.				
Steps	User clicks Sign Up. User enters a valid password in “Password” but a different value in “Confirm Password.” User clicks Sign Up.				
Expected	Error message indicates the passwords do not match; account is not created.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_1.6 – Sign Up with Existing Email	Category	Functional ▾	Severity	Major ▾
Objective	Check that the system prevents duplicate accounts with the same email.				
Steps	User clicks Sign Up. User enters an email that already exists in the system. User fills in other fields and clicks Sign Up.				
Expected	Error message “Email already exists” (or similar); account creation is blocked.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_2.1 – Log In with Valid Credentials	Category	Functional ▾	Severity	Major ▾
Objective	Ensure that users can log in with the correct email/password.				
Steps	User opens the extension. User clicks Sign In. User enters a valid email and password. User clicks Sign In.				
Expected	User is redirected to the MainMainPage				
Result	Pass				
Date	01.03.2025				

Test ID	TC_2.2 – Log In with Invalid Password	Category	Functional ▾	Severity	Major ▾
Objective	Verify incorrect passwords are rejected.				
Steps	User opens the extension. User clicks Sign In. User enters the correct email but an incorrect password. User clicks Sign In.				
Expected	An error message is displayed (e.g., “Invalid password”); login fails.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_2.3 – Log In with Non-Existent Email	Category	Functional ▾	Severity	Major ▾
Objective	Check that users cannot log in using an unregistered email.				
Steps	User opens the extension. User clicks Sign In. User enters an email not found in the database. User clicks Sign In.				
Expected	Error message “Email not found” (or similar) is displayed.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_2.4 – Log In with Empty Fields	Category	Functional ▾	Severity	Major ▾
Objective	Ensure the system requires both email and password.				
Steps	User opens the extension, clicks Sign In. User leaves email or password blank. User clicks Sign In.				
Expected	System displays an error indicating missing credentials.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_3.1 – Standard Logout	Category	Functional ▾	Severity	Major ▾
Objective	Ensure that clicking logout signs the user out.				
Steps	User clicks the logout icon.				
Expected	User is signed out and redirected to the login page (or sees a logged-out state).				
Result	Pass				
Date	01.03.2025				

Test ID	TC_3.2 – Logout via Inactivity	Category	Functional	Severity	Minor
Objective	Check if the system auto-logs out a user after inactivity.				
Steps	User logs in, then remains idle for the configured time limit.				
Expected	System logs the user out automatically, requiring re-login.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_4.1 – Analyze a Valid Article Page	Category	Functional	Severity	Critical
Objective	Verify that analyzing a valid news/article page works.				
Steps	User clicks Analyze Current Page.				
Expected	System displays reliability score, analysis summary, similar articles.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_4.2 – Analyze a Non-Article Page or Invalid Content	Category	Functional	Severity	Major
Objective	Ensure the system handles invalid pages				

Steps	User is on a page that is not a typical article (e.g., a blank page). User clicks Analyze Current Page.
Expected	System either shows an error (“Unable to analyze”) or minimal results.
Result	Pass
Date	01.03.2025

Test ID	TC_5.1 – Analyze Valid URL	Category	Functi… ▾	Severity	Critical ▾
Objective	Verify user can input a URL to be analyzed with the same reliability features.				
Steps	User enters a valid URL in the input box. User clicks Analyze URL.				
Expected	System displays reliability score, similar articles, etc.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_5.2 – Analyze Invalid or Empty URL	Category	Functi… ▾	Severity	Major ▾
Objective	Check system response to an invalid or no URL.				
Steps	User leaves the URL box empty or types a malformed URL (e.g., http://notvalid). User clicks Analyze URL.				

Expected	Error message prompts user that the URL is invalid, or to fill in the input box.
Result	Pass.
Date	01.03.2025

Test ID	TC_6 – View More Details for Valid Analysis	Category	Functi…	Severity	Major
Objective	Verify that clicking “Show more details” loads additional information about the article.				
Steps	After the article is analyzed, the user clicks Show More Details.				
Expected	System shows extra details (stats, references, etc.).				
Result	Pass				
Date	01.03.2025				

Test ID	TC_7.1 – Change Plan to Premium	Category	Functi…	Severity	Major
Objective	Ensure a user can upgrade from free to premium.				
Steps	User clicks the profile icon. User selects “Premium” plan.				
Expected	Plan changes to premium; user sees updated subscription status.				

Result	Pass
Date	01.03.2025

Test ID	TC_7.2 – Change Plan to Enterprise	Category	Functi…	Severity	Major
Objective	Verify a user can change from premium/free to enterprise.				
Steps	User clicks the profile icon. User selects “Enterprise” plan.				
Expected	User subscription changes to enterprise plan.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_7.3 – Change Plan to Free	Category	Functi…	Severity	Major
Objective	Check a user can revert back to free.				
Steps	User clicks the profile icon. Selects “Free” plan.				
Expected	Subscription changes to free; possibly showing a downgrade confirmation.				
Result	Pass				

Date	01.03.2025

Test ID	TC_7.4 – Change Plan with Payment Failure	Category	Functi…	Severity	Major
Objective	See what happens if payment for a paid plan fails.				
Steps	User selects a paid plan. Payment gateway returns a failure or declines the card.				
Expected	System notifies the user of payment failure; the plan remains unchanged.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_8.1 – Change Password Successfully	Category	Functi…	Severity	Minor
Objective	User can change their current password to a new one.				
Steps	User clicks the profile icon. User selects Change Password. User enters the new password and confirms it correctly. User clicks Change Password button.				
Expected	Password updates; user sees success message.				

Result	Pass
Date	01.03.2025

Test ID	TC_8.2 – Change Password with Mismatched Confirmation	Category	Functi…	Severity	Minor
Objective	Ensure mismatched new passwords are detected.				
Steps	User enters a new password but a different confirmation password. User clicks Change Password.				
Expected	Error message: “Passwords do not match.”				
Result	Pass				
Date	01.03.2025				

Test ID	TC_8.3 – Change Password with Blank Fields	Category	Functi…	Severity	Minor
Objective	Check that all fields are required.				
Steps	User leaves the new password or confirm password field empty. User clicks Change Password.				
Expected	Error message: “Please fill all required fields.”				

<table>
<tr><td>Result</td><td>Pass</td></tr>
<tr><td>Date</td><td>01.03.2025</td></tr>
</table>

<table>
<tr><td>Test ID</td><td>TC_9 – View Dashboard</td><td>Category</td><td>Functi…</td><td>Severity</td><td>Major</td></tr>
<tr><td>Objective</td><td colspan="5">Check that clicking “Your Dashboard” shows correct stats.</td></tr>
<tr><td>Steps</td><td colspan="5">User logs in.
User clicks MainMenuPage → “Your Dashboard” (or Profile → Dashboard).</td></tr>
<tr><td>Expected</td><td colspan="5">Dashboard displays daily limit, # of articles analyzed, average reliability score, total articles.</td></tr>
<tr><td>Result</td><td colspan="5">Pass</td></tr>
<tr><td>Date</td><td colspan="5">01.03.2025</td></tr>
</table>

<table>
<tr><td>Test ID</td><td>TC_10 – View History</td><td>Category</td><td>Functi…</td><td>Severity</td><td>Major</td></tr>
<tr><td>Objective</td><td colspan="5">Check that clicking “Your Dashboard” shows correct stats.</td></tr>
<tr><td>Steps</td><td colspan="5">User logs in.
User clicks MainMenuPage → “Your Dashboard” (or Profile → Dashboard).</td></tr>
<tr><td>Expected</td><td colspan="5">Dashboard displays daily limit, # of articles analyzed, average reliability score, total articles.</td></tr>
</table>

Result	Pass
Date	01.03.2025

Test ID	TC_11 – Send Report	Category	Functi…	Severity	Major
Objective	Confirm a user can file a “mistake” report.				
Steps	User clicks Report Icon or Report Mistake. User selects “Mistake” as the report type. User enters a message. User clicks Send Report.				
Expected	Report is sent to admins.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_12.1 – Forgot Password with Valid Email	Category	Functi…	Severity	Major
Objective	Ensure the user can reset their password using a correct email and code.				
Steps	User clicks Sign In → Forgot Password link. User enters their valid email and clicks Send Verification Code. User receives the code by email, enters it, and sets a new password. User confirms new password.				

Expected	Password is updated; user can log in with the new password.
Result	Pass
Date	01.03.2025

Test ID	TC_12.2 – Forgot Password with Unregistered Email	Category	Functi…	Severity	Minor
Objective	Verify the system rejects invalid/unregistered emails for password reset.				
Steps	User clicks Forgot Password. User enters an email not in the system. User clicks Send Verification Code.				
Expected	Error message: “Email not found” or similar.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_12.3 – Forgot Password with Invalid Verification Code	Category	Functi…	Severity	Major
Objective	Check that an incorrect code is not accepted.				

Steps	User enters a verification code that is wrong or expired. User tries to set a new password.
Expected	Error message indicates invalid or expired code.
Result	Pass
Date	01.03.2025

Test ID	TC_12.4 – Forgot Password with Mismatched New Passwords	Category	Functi…	Severity	Major
Objective	Ensure new password and confirm password must match.				
Steps	User enters a valid verification code. User types a new password but a different confirm password. User clicks Submit.				
Expected	Error message “Passwords do not match.”				
Result	Pass				
Date	01.03.2025				

Test ID	TC_13 – View Credibility after Analyzing	Category	Functi…	Severity	Critical

Objective	User can see if a website is “credible,” “mixed,” or “uncredible.”
Steps	User clicks Analyze Current Page or inputs URL. After results load, user clicks More Details.
Expected	System displays credibility status (“credible,” “mixed,” or “uncredible”).
Result	Pass
Date	01.03.2025

Test ID	TC_14.1 – Similarity Detection Module	Category	Functional	Severity	Critical
Objective	Verify the similarity detection module correctly identifies text similarity.				
Steps	Provide two identical text inputs to the module				
Expected	Output = 1 (identical texts).				
Result	Pass				
Date	01.03.2025				

Test ID	TC_14.2 – Similarity Detection Module	Category	Functional	Severity	Critical
Objective	Verify the similarity detection module correctly identifies text similarity.				

Steps	Provide two completely different with no semantic similarity text inputs to the module.
Expected	Output = 0 (no similarity).
Result	Pass
Date	01.03.2025

Test ID	TC_14.3 – Similarity Detection Module	Category	Functional	Severity	Critical
Objective	Verify the similarity detection module correctly identifies text similarity.				
Steps	Provide two partially similar text inputs to the module.				
Expected	Output = a value between 0 and 1 (partial similarity).				
Result	Pass				
Date	01.03.2025				

Test ID	TC_15.1 – Objectivity Classification Model	Category	Functional	Severity	Critical
Objective	Verify the objectivity classification model correctly classifies text as objective or subjective.				
Steps	Provide a factual news article (e.g., "The population of New York is 8.5 million")..				

Expected	Output = "Objective".
Result	Pass
Date	01.03.2025

Test ID	TC_15.2 – Objectivity Classification Model	Category	Functional	Severity	Critical
Objective	Verify the objectivity classification model correctly classifies text as objective or subjective.				
Steps	Provide an opinion piece (e.g., "New York is the best city in the world").				
Expected	Output = "Subjective".				
Result	Pass				
Date	01.03.2025				

Test ID	TC_16.1 – Political Bias Classification Model	Category	Functional	Severity	Critical
Objective	Verify the political bias classification model correctly identifies bias as left, center, or right.				

Steps	Provide a text with left-leaning political bias (e.g., "The government should increase taxes on the wealthy to fund social programs").
Expected	Output = "Left".
Result	Pass
Date	01.03.2025

Test ID	TC_16.2 – Political Bias Classification Model	Category	Functional	Severity	Critical
Objective	Verify the political bias classification model correctly identifies bias as left, center, or right.				
Steps	Provide a text with right-leaning political bias (e.g., "Lower taxes for businesses will stimulate economic growth").				
Expected	Output = "Right".				
Result	Pass				
Date	01.03.2025				

Test ID	TC_16.3 – Political Bias Classification Model	Category	Functional	Severity	Critical

Objective	Verify the political bias classification model correctly identifies bias as left, center, or right.
Steps	Provide a politically neutral text
Expected	Output = "Center".
Result	Pass
Date	01.03.2025

Test ID	TC_17.1 – Reliability Score Model	Category	Functional ▾	Severity	Critical ▾
Objective	Verify the reliability scoring model correctly assigns a reliability score between 0 and 1.				
Steps	Input a text article to the model and check if the model outputs a reliability score.				
Expected	A reliability score between 0-1 should be retrieved.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_17.2 – Reliability Score Model	Category	Functional ▾	Severity	Critical ▾
Objective	Verify the reliability scoring model correctly assigns a reliability score between 0 and 1.				

Steps	Input a fake news article and receive an output
Expected	A reliability score between 0-0.5 should be retrieved.
Result	Pass
Date	01.03.2025

Test ID	TC_17.3 – Reliability Score Model	Category	Functional	Severity	Critical
Objective	Verify the reliability scoring model correctly assigns a reliability score between 0 and 1.				
Steps	Input a real news article and receive an output				
Expected	A reliability score between 0.5-1 should be retrieved.				
Result	Pass				
Date	01.03.2025				

Test ID	TC_18.1 – Edge Case Testing for Similarity Detection Model	Category	Edge …	Severity	Major
Objective	Verify the similarity detection model handles edge cases correctly.				
Steps	Provide two texts with only stopwords (e.g., "the and is").				
Expected	Output = 1 (stopwords are considered identical)				

Result	Pass
Date	01.03.2025

Test ID	TC_18.2 – Edge Case Testing for Similarity Detection Model	Category	Edge … ▾	Severity	Major ▾
Objective	Verify the similarity detection model handles edge cases correctly.				
Steps	Provide one text and an empty string as the second input.				
Expected	Output = 0 (no similarity with an empty string)				
Result	Pass				
Date	01.03.2025				

Test ID	TC_19 – Edge Case Testing for Objectivity Classification Model	Category	Edge … ▾	Severity	Major ▾
Objective	Verify the objectivity classification model handles ambiguous or mixed texts.				
Steps	Provide a text that mixes facts and opinions (e.g., "The population of New York is 8.5 million, and it's the best city in the world").				
Expected	Output = "Subjective" (since the text contains opinions).				
Result	Pass				

Date	01.03.2025

Test ID	TC_20 – Edge Case Testing for Political Bias Classification Model	Category	Edge …	Severity	Major
Objective	Verify the political bias classification model handles neutral or ambiguous texts.				
Steps	Provide a text with no political context (e.g., "The weather today is sunny").				
Expected	Output = "Center" (neutral).				
Result	Pass				
Date	01.03.2025				

Test ID	TC_21 – Integration of All Models	Category	Integra…	Severity	Critical
Objective	Verify all models work together seamlessly to provide a comprehensive fact-checking output.				
Steps	Provide a news article as input. Run the article through all models. ● Similarity detection ● Objectivity classification ● Political bias classification ● Reliability scoring				

Expected	Similarity detection output = value between 0 and 1. Objectivity classification output = "Objective" or "Subjective". Political bias classification output = "Left", "Center", or "Right". Reliability scoring output = value between 0 and 1.
Result	Pass
Date	01.03.2025

Test ID	TC_22 – Claim Extraction	Category	Functional ▾	Severity	Critical ▾
Objective	Ensure that claim extraction is complete				
Steps	Input a news article text.				
Expected	At least one claim must be extracted.				
Result	Pass				
Date	01.03.2025				

Test ID	NFR-1 – Clear Visual Indicators	Category	Non-f… ▾	Severity	Major ▾
Objective	Ensure color-coded scoring and political compass graphics are visible and understandable.				
Steps	User opens an article analysis page in the extension.				

	Inspect that the reliability score is color-coded (e.g., green=credible, yellow=mixed, red=uncredible). Check for a political compass graphic or textual indicator.
Expected	Colors, icons, or textual labels are clear and consistent
Result	Pass
Date	01.03.2025

Test ID	NFR-2 – Browser Compatibility	Category	Comp…	Severity	Major
Objective	The system is accessible on all major Chromium browsers (Google Chrome, Opera, Microsoft Edge and Brave) as well as Mozilla Firefox and Safari.				
Steps	Use the extension				
Expected	No broken layouts or overlapping elements. All critical functionality remains accessible at different browsers.				
Result	Pass except for Safari.				
Date	01.03.2025				

Test ID	NFR-3 – Mobile UI Responsiveness & Usability	Category	Usability	Severity	Major

Objective	Validate that the mobile app’s interface is optimized for smaller screens and different orientations (portrait/landscape).
Steps	Open the app on various device sizes (phone and tablet). Navigate through menus, open analysis results, and check any interactive elements (e.g., text fields, buttons). (Optional) Rotate the device from portrait to landscape.
Expected	No UI elements are cut off or overlapping. Buttons/fields are appropriately scaled and easy to tap. Navigation remains logical in both orientations.
Result	Pass
Date	01.03.2025

Test ID	PERF-1 – Response Time for Browser Extension	Category	Perfor…	Severity	Critical
Objective	Verify both browser extension and mobile app respond within acceptable time limits for all user actions.				
Steps	Perform article analysis request using the browser extension. Perform article analysis request using the mobile app.				
Expected	Response time ≤ 5 seconds for both browser extension and mobile app.				
Result	Pass				
Date	01.03.2025				

Test ID	M-1 – Complete Mobile App	Category	Functi… ▾	Severity	Critical ▾
Objective	Ensure that core features (article analysis, reliability scoring, political compass, account management, etc.) are available and function correctly on the mobile app, matching the desktop extension.				
Steps	Launch the mobile app on a phone or tablet. Log in. Access and use each major feature (e.g., article analysis, viewing reliability scores, changing subscription plan).				
Expected	All primary features that exist on the browser extension are also present in the mobile app and function without major differences or missing steps.				
Result	Pass				
Date	01.03.2025				

Test ID	M-2 – Log in to the Mobile App	Category	Functi… ▾	Severity	Critical ▾
Objective	Ensure that login works on the mobile application, and the token is generated.				
Steps	Open the app. Enter the email and the password. Press the login button.				
Expected	User is navigated to the homepage. Token is generated.				

Result	Pass
Date	08.03.2025

Test ID	M-3 – Log out of the Mobile App	Category	Functi…	Severity	Critical
Objective	Ensure that the user logs out upon token expiry.				
Steps	Open the app. Log in. Wait for 1 hour (token expiry).				
Expected	User is automatically logged out.				
Result	Pass				
Date	08.03.2025				

Test ID	M-5 – More Details page of Article Analysis in Mobile App	Category	Functi…	Severity	Critical
Objective	Ensure that the user is automatically redirected to the article page after they analyze the article on the mobile app.				
Steps	Open the app. Login				

	Press the ‘Search’ button that is on the bottom-right of the page. Enter the article url. Press the ‘Analyze’ button.
Expected	User is navigated to the article page.
Result	Pass
Date	08.03.2025

6. Maintenance Plan and Details

To ensure CheckMate remains accurate, secure, and efficient over time, we have developed a comprehensive maintenance strategy addressing all system components. Our plan focuses on keeping models, dependencies, and databases updated while continuously incorporating user feedback for ongoing improvement.

6.1 Model Maintenance

Our machine learning models require regular updates to maintain and improve their accuracy:

- **Regular Retraining Schedule**: We will retrain our core machine learning models (subjectivity classifier, political bias classifier, and reliability score model) on a monthly basis to incorporate new data patterns and prevent model drift. This process will use our continuously expanding dataset of analyzed articles and user feedback.
- **API Version Management**: As new versions of Google Text Embedding API and TensorFlow-based BERT models are released, we will implement a two-phase integration approach:

1. Testing in a controlled environment to measure performance impacts

    2. Gradual rollout to production, ensuring backward compatibility
- **Performance Benchmarking**: We will maintain a curated test dataset of verified articles with known reliability characteristics to validate model performance after each update cycle.

6.2 Dependency Management

The software dependencies critical to our system will be routinely monitored and updated:

- **Backend Framework Updates**: Our Flask, SQLAlchemy, and other Python dependencies will undergo monthly update assessments based on the project's compatibility matrix. Version upgrades will be tested in a staging environment before deployment.
- **Browser Compatibility**: We will update our extension to always be in line with new browser updates, ensuring the application will be stable across multiple browsers.
- **Library Version Control**: A dependency lockfile will be maintained to ensure consistent deployments, with weekly automated checks for security vulnerabilities using tools like safety and Snyk.

6.3 Database Maintenance

Our PostgreSQL database requires regular optimization to manage the growing volume of data:

- **Data Retention Policy**: We will implement a tiered data retention strategy:
    - Analysis results will be maintained for 12 months
    - User search history for 6 months
    - Raw extraction data for 3 months
- **Index Optimization**: Query patterns will be analyzed quarterly to optimize database indexes, with special attention to the article_searches and similar_articles tables that experience the highest query volume.

- **Schema Evolution**: As new features are developed, database migrations will be deployed during scheduled maintenance windows using a version-controlled migration framework to ensure data integrity.

6.4 User Feedback Integration

User feedback is a critical component for improving system accuracy:

- **Feedback Collection**: The report submission feature will categorize user feedback into specific improvement areas (reliability scoring issues, bias detection problems, similarity calculation errors).
- **Weight Adjustment**: Based on aggregated feedback patterns, we will adjust the weights in our reliability model quarterly after statistical validation of proposed changes.
- **Validation Process**: All modifications resulting from user feedback will undergo A/B testing with a subset of production traffic before full deployment to verify improvements.

6.5 System Monitoring

Proactive monitoring will help identify and resolve issues before they impact users:

- **Performance Metrics**: We will track key performance indicators including:
    - Average response time for article analysis
    - Model inference latency
    - API call success rates
    - Error frequency by component
- **Alerting System**: Automated alerts will be configured for anomalies in system performance, with escalation paths defined for critical issues.
- **Usage Analytics**: Regular reports will analyze user interaction patterns to identify high-traffic periods and optimize resource allocation.

6.6. Scheduled Maintenance

Regular maintenance activities will be coordinated to minimize disruption:

- **Maintenance Windows**: Routine updates will be scheduled during periods of lowest user activity, typically late night or early morning hours.
- **Update Deployment**: A canary deployment approach will be used for significant changes, with initial rollout to a small percentage of traffic before full implementation.
- **Communication**: Users will be notified of scheduled maintenance through the extension interface and email (for registered users) with appropriate lead time.

6.7. Documentation Updates

Maintaining accurate documentation is essential for system sustainability:

- **Internal Documentation**: Technical documentation will be updated in lockstep with code changes to ensure development knowledge is preserved.
- **Release Notes**: Each maintenance cycle will be accompanied by detailed release notes explaining improvements, bug fixes, and any changes to user experience.
- **Knowledge Base**: A public-facing knowledge base will be maintained to answer common questions and provide transparency about system operation.

Through this structured maintenance approach, we aim to continuously improve CheckMate's accuracy and performance while maintaining system stability and user trust. Each component of the plan includes clear ownership, timelines, and success metrics to ensure effective execution.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

Effect Level	Effect
Public Health	1 Minimal — Misinformation can indirectly influence health behavior in some news topics.
Public Safety	2 Low — In crisis events, false information could lead to panic or misinformed reactions.
Public Welfare	3 Moderate — Helps combat disinformation that may distort civic awareness and social trust.
Global Factors	5 High — Requires internationalization, multi-language NLP support, and cross-cultural bias handling.
Cultural Factors	4 Significant — Needs culturally sensitive phrasing, adaptable UI, and localized moderation.
Social Factors	5 High — Must be transparent in AI decision-making to promote user trust and fight polarization.
Environmental Factors	3 Medium — Should optimize cloud inference and embedding usage to minimize carbon footprint.
Economic Factors	6 Very High — Critical to keep deployment costs low (e.g., $1/user/month) for broad accessibility.

7.1.1 Constraints

7.1.1.1 Implementation Constraints

- The browser extension is compatible with all major Chromium browsers (Google Chrome, Opera, Microsoft Edge and Brave) as well as Mozilla Firefox [5].
- GitHub and Jira are used to track the deadlines, issues, and the codebase.
- HTML and CSS frameworks are used for frontend development.
- Python is used for machine learning (ML) development and backend development.
- PostgreSQL hosted by Amazon RDS server is used for the database system.
- Google Cloud server is used for ML model and backend deployment.

7.1.1.2 Economic Constraints

- Our project requires several external libraries, frameworks, and models. Therefore, our group has opted to use as many open-source frameworks as possible.
- Google Vision API [14], Google Custom Search API [11], ML model and backend deployment on Amazon EC2 servers use a paid plan after the free tier parameters are reached.

7.1.1.3 Ethical Constraints

- All interactions and data collected from the users are handled within data protection law General Data Protection Regulation (GDPR).
- The users are informed about the application's limitations and scope before registration.
- The system utilizes personal data and storing said data in the database hosted on Amazon RDS.
- No unnecessary user data will be collected.
- The system will clearly communicate to the user about its shortcomings.
- The system will clearly communicate reasons for reliability scores of the news articles and the reasons for credibility scores for news sources.
- The system will suggest trustworthy sources to the user about the news article that they are searching for.

- The system is designed to ensure objectivity and fairness, actively compensating for any biased outcomes to provide accurate labeling of news articles and news sources.

7.1.1.4 Language Constraints

- The system will work on news articles and new sources in English because of the lack of labeled Turkish news article datasets.

7.1.2 Standards

- The system will abide by the European Fact-Checking Standards Network (EFCSN) [16] as a benchmark in fact-checking news articles.
- The system will abide by Google Cloud Vision API Terms of Service, Bing Search API License Agreement, and other third-party API usage policies.
- The system will abide to General Data Protection Regulation (GDPR) [17]

7.2 Ethics and Professional Responsibilities

The CheckMate project team is committed to upholding ethical standards and professional responsibilities. Key considerations include:

- Protecting user privacy and data security in compliance with GDPR and other relevant regulations.
- Maintaining transparency about the system's capabilities, limitations, and potential biases.
- Ensuring the credibility assessments are fair, unbiased, and not used to suppress legitimate political views.
- Addressing user concerns and feedback in a timely and respectful manner.
- Continuously improving the system's accuracy and reliability through rigorous testing and updates.

7.3 Teamwork Details

7.3.1 Contributing and functioning effectively on the team to establish goals, plan tasks, and meet objectives

Our team collaborates using a combination of Agile-inspired methodologies, structured task management, and frequent communication channels to ensure that everyone contributes effectively to the project:

7.3.1.1 Project Management with JIRA

We use JIRA as our primary tool for planning and tracking tasks. Each sprint is broken down into subtasks, which are then assigned to individual team members. JIRA's board view provides transparency into task status (To Do, In Progress, Done) and helps us monitor deadlines, prioritize workload, and spot any bottlenecks.

7.3.1.2 Regular Communication and Meetings

We maintain a WhatsApp group for rapid, day-to-day communication, allowing team members to quickly share updates, ask questions, and coordinate ad-hoc tasks. In addition to ongoing online communication, we hold three weekly meetings to discuss progress, align on upcoming goals, and resolve any challenges. .

7.3.1.3 Role Division and Collaboration

Batu and Pelin were responsible for architecting and implementing the backend infrastructure of CheckMate. Their work included:

- Designing and developing RESTful APIs using Flask, ensuring consistent, secure, and scalable endpoints for frontend communication.
- Implementing JWT-based authentication and route protection, as well as email verification logic.
- Managing database schema design and optimization using SQLAlchemy with a PostgreSQL backend.

- Developing modules for article scraping, similarity computation, domain credibility validation, and result aggregation.
- Orchestrating API calls to external services (e.g., Google Custom Search, Vertex AI) and handling concurrency using ProcessPoolExecutor.
- Ensuring robust error handling, transaction rollbacks, and performance tuning of core endpoints.

Ayberk led the development of all user-facing components, including:

- Designing and implementing the browser extension UI, compatible with Chrome, Edge, and Firefox.
- Building the mobile application using CapacitorJS, supporting both iOS and Android platforms.
- Integrating frontend components with backend APIs, handling asynchronous data fetching, token management, and real-time UI updates.
- Implementing user interaction features such as login/registration, result visualization, color-coded reliability indicators, and profile/dashboard management.
- Ensuring a responsive and accessible user experience across platforms through structured component-based design.

Ipek and Efe developed and deployed the core intelligence behind CheckMate’s analysis pipeline. Their responsibilities included:

- Training and fine-tuning models for:
    - Political Bias Classification using Transformer-based architectures.
    - Text and Title Objectivity Analysis via custom RNNs and subjectivity scoring techniques.
    - Reliability Scoring using a Multi-Layer Perceptron (MLP) built with scikit-learn.

- Engineering input features, such as similarity metrics, extremism scores, and grammatical quality indicators.
- Deploying models as independent FastAPI microservices on Google Cloud Run, ensuring low-latency inference at scale.

- Integrating Google Vertex AI for text embeddings.
- Conducting evaluation, model validation, and system integration testing to ensure consistency with backend requirements.

Despite these primary roles, we encourage knowledge sharing and cross-functional support. If someone encounters a roadblock, teammates from any domain can step in to offer insights.

7.3.1.4 Coordination and Accountability

Each member is responsible for updating JIRA tasks with progress notes and shifting status as they move from development to testing. Even if there is a role division each member gets to work on a different team if another team needs their help.

By combining these practices, our team maintains clear communication, well-defined responsibilities, and a supportive environment that enables us to efficiently develop CheckMate’s backend, frontend, and machine learning models.

7.3.2 Helping creating a collaborative and inclusive environment

Each team member takes the lead on a specific core component, but that doesn't prevent us from assisting one another with coding challenges. Whenever someone encounters an issue while coding we all work on it together by sharing the code with the issue among each other, ensuring that everyone is informed and can analyze the code to help find a solution.

Additionally, we use WhatsApp and Google meet to discuss problems and share our planned solutions, keeping the entire team updated. We recognize that this is a collaborative project, and although tasks are divided among us, we all share the responsibility of delivering a fully developed application.

7.3.3 Taking lead role and sharing leadership on the team

Our team embraces a domain-based leadership model, where individuals lead when they have expertise in a specific area. For example, someone well-versed in mobile app development guided that portion of the project, while another person with deep ML knowledge led model selection and training. However, all major decisions still go through a collaborative process, ensuring that everyone's perspectives are considered.

Domain leads also act as mentors, sharing knowledge and providing hands-on guidance to team members less familiar with their area of expertise. This approach fosters ownership, as each member takes initiative in their specialty. It also keeps us agile when new challenges arise (optimizing the ML pipeline), the respective domain lead steps forward, with others offering support.

Overall, this flexible leadership style ensures that each person can leverage their strengths, while the team benefits from a shared sense of responsibility and balanced decision-making.

7.3.4 Meeting objectives

- Three-day weekly cycle: We meet every Tuesday, Thursday and Saturday to plan, build and wrap up our goals.
- Set clear targets: At the start of each meeting we pick 1–3 concrete objectives ("Add reliability endpoint," "Improve UI response time").
- Work together: Tasks get split on the spot and we pair-program or troubleshoot in real time—no one works in isolation.
- Quick demo & sign-off: By the end of the daily session we do a 5-minute demo to verify each objective is complete, then close the loop until the next meeting.

7.4 New Knowledge Acquired and Applied

Over the course of building CheckMate, our team developed and applied a wide range of practical technical skills spanning backend development, web scraping, cloud integrations, performance optimization, and full-stack connectivity. Below is a breakdown of key competencies we acquired and how we utilized them in the system:

7.4.1 Flask & API Design

We gained deep proficiency in Flask and modern RESTful API design patterns. Our backend services were structured around clear route definitions, robust request validation, and secure access. We implemented JWT-based authentication to protect sensitive endpoints and introduced CORS handling to support cross-origin requests from both browser extensions and mobile clients. All API responses were returned in well-structured JSON format, and we implemented consistent HTTP error handling with descriptive status codes and user-friendly error messages. We also improved our ability to log exceptions and roll back transactions cleanly, ensuring backend reliability and transparency.

7.4.2 Web Scraping Techniques

To power our article analysis engine, we built a resilient web scraping pipeline combining BeautifulSoup-based heuristics with headless browser automation using undetected-chromedriver. This dual-layer approach allowed us to extract content from both static and dynamically rendered pages. We refined our logic to identify and isolate the main article content while filtering out ads, navigation headers, thumbnail blocks, and unrelated boilerplate elements. The scraper intelligently prioritized structured HTML elements and fallback metadata, making it highly adaptable across different publisher formats.

7.4.3 Cloud & AI Integrations

We integrated several external services to enable CheckMate's AI-powered features. This included connecting to the Google Custom Search API for reverse article lookup, Google Vertex AI for semantic embeddings, and Google Cloud Vision API (early prototype) for experimental image-based content validation. These integrations taught us how to correctly structure API payloads, interpret complex nested JSON responses, and handle edge cases such as timeouts, rate limits, and malformed data. We learned how to integrate asynchronous API calls into a real-time inference pipeline with meaningful error handling.

7.4.4 Parallelism & Performance Optimization

Performance was crucial to support near-instant feedback for users. We employed Python's ProcessPoolExecutor to parallelize the scraping and analysis of up to 10 reverse-searched articles, significantly reducing total response time. We also implemented logic to clamp, validate, and normalize the similarity scores to ensure they stayed within acceptable backend ranges. This attention to numerical stability prevented scoring anomalies and improved the reliability of downstream inferences. We learned how to balance concurrency with system resource constraints and maintain consistent throughput under load.

7.4.5 Database & ORM Proficiency

We deepened our knowledge of SQLAlchemy, focusing on relational mapping, ORM relationships, and transactional workflows. We implemented one-to-many and many-to-one relationships to link users, articles, and analysis logs effectively. We also resolved practical issues like VARCHAR truncation, default value handling, and data type mismatches. Our backend services supported bulk inserts and batch updates, helping us maintain performance and avoid redundant computation. We also developed logic to map domain names to credibility values stored in the trusted_websites table, enhancing source validation.

7.4.6 Frontend–Backend Integration

We established seamless communication between the backend Flask API and frontend clients, especially our Chrome extension. We used fetch() for API calls, maintained session state with localStorage, and dynamically updated the DOM to display real-time analysis results. We added user-friendly elements like color-coded reliability scores, tooltips, and expandable breakdowns to communicate credibility, bias, objectivity, and grammar insights. This full-stack wiring gave us hands-on experience designing data flows between disparate components and managing cross-platform consistency.

7.4.7 DevOps & Resilience Engineering

To support long-term maintainability, we learned key DevOps principles, including environment management, observability, and graceful degradation. We managed sensitive credentials and API keys using python-dotenv, enforced retry logic on unstable external services, and introduced timeouts and fallback responses to prevent the system from crashing during third-party outages. Logging was centralized and categorized to differentiate critical failures from recoverable warnings, making debugging and post-mortem analysis more efficient.

8. Conclusion and Future Work

As we look ahead beyond the initial scope of our senior design project, we recognize the vast potential for expanding CheckMate's capabilities to deliver even deeper insights into news content. Two promising directions we have already begun to explore are visual analysis integration and the implementation of Retrieval-Augmented Generation (RAG) [23] for advanced article summarization and title assessment.

Firstly, the addition of visual analysis will allow CheckMate to assess the credibility and emotional impact of images embedded within articles. Since visual media plays a significant role in shaping user perception and can often be manipulated to mislead or exaggerate, this enhancement will provide users with a more holistic understanding of article framing. By integrating image classification, facial expression analysis, and reverse image search pipelines—potentially leveraging services like Google Cloud Vision API—we aim to flag emotionally charged or out-of-context visuals that may distort narrative neutrality.

Secondly, implementing a RAG-based summarization module represents a major advancement in how CheckMate evaluates content. Unlike traditional extractive summarization methods, RAG enables us to dynamically query a curated evidence corpus and generate summaries that contextualize the article in relation to broader verified knowledge. This will not only improve the accuracy and informativeness of article summaries, but also significantly enhance title objectivity analysis, as titles can then be compared against a fact-aware, contextually grounded summary to detect sensationalism or misrepresentation. RAG will also strengthen our reverse search interpretation by grounding retrieved content more explicitly in known facts.

These forward-looking enhancements reinforce our vision for CheckMate as not just a fact-checking tool, but a comprehensive information literacy platform. By expanding into multimodal analysis and retrieval-aware language generation, we aim to empower users with richer, more nuanced assessments of online news, making CheckMate an indispensable companion in the pursuit of truth in digital media.

9. Glossary

API: An API, or Application Programming Interface, is a collection of rules and protocols that allow software applications to interact and share data, features, or functionality. API communication can be understood as an exchange of requests and responses between a client and a server. The client is the application sending the request, while the server processes the request and sends back a response. The API acts as the intermediary, facilitating this connection between the two. [18]

NLP: Natural Language Processing (NLP) is a branch of computer science and artificial intelligence focused on enabling computers to comprehend human language. It combines computational linguistics, which examines the mechanics of language, with statistical methods, machine learning, and deep learning models. These techniques equip computers to analyze and interpret text or speech data, understanding their overall meaning, as well as the speaker's or writer's intentions and emotions. [19]

ML: Machine learning (ML) is a branch of artificial intelligence (AI) and computer science that focuses on using data and algorithms to enable AI to imitate the way that humans learn, gradually improving its accuracy. [20]

URL: The location of a webpage or file on the Internet. [21]

MLP: Multilayer perceptron (MLP) is a type of artificial neural network used in machine learning to make predictions. It is made up of layers of connected units called neurons. These neurons are arranged in three main parts: an input layer that receives the data, one or more hidden layers that process the data, and an output layer that gives the result. Each neuron passes information to the next layer after applying a simple calculation, often using a function that adds non-linearity. MLPs learn by adjusting the strength of the connections between neurons. This is done during training using a method called backpropagation, which helps the network reduce errors and make better predictions over time. [22]

RAG: Retrieval-Augmented Generation is a is a system that improves an AI model's effectiveness by linking it to additional reference databases. [23]

10. References

[1] Meta, "How fact-checking works," Meta Transparency Center. [Online]. Available: https://transparency.meta.com/en-us/features/how-fact-checking-works/. [Accessed: 16-Nov-2024].

[2] X Help Center, "Community Notes," X, [Online]. Available: https://help.x.com/en/using-x/community-notes. [Accessed: Mar. 8, 2025].

[3] Knight Science Journalism, "Fact-checking science journalism: How to make sure your stories are true," KSJ Handbook. [Online]. Available: https://ksjhandbook.org/fact-checking-science-journalism-how-to-make-sure-your-stories-are-true/the-fact-checking-process/. [Accessed: 16-Nov-2024].

[4] J. Waterson, "Internet overtakes TV as UK's most popular news source for first time," The Guardian, 10-Sep-2024. [Online]. Available: https://www.theguardian.com/media/article/2024/sep/10/internet-tv-uk-most-popular-news-source-first-time. [Accessed: 16-Nov-2024].

[5] Mihir J, "Browser Comparison Finale Chromium-based Browsers", Medium. [Online]. Available:https://medium.com/@mihirgrand/browser-comparison-finale-chromium-based-browsers-2b6063e74165. [Accessed: 16-Nov-2024].

[6] Werkzeug Security. "Werkzeug," PyPI. [Online]. Available: https://pypi.org/project/Werkzeug/. [Accessed: 08-Nov-2024].

[7] T. Tiangolo, "FastAPI: Modern, Fast (high-performance), web framework for building APIs with Python," [Online]. Available: https://fastapi.tiangolo.com/. [Accessed: 08-Mar-2025].

[8] Google Cloud, "Vertex AI Embeddings API," [Online]. Available:

https://cloud.google.com/vertex-ai/docs/generative-ai/embeddings/overview. [Accessed: 08-Mar-2025].

[9] Amazon Web Services, "Amazon RDS," AWS, [Online]. Available: https://aws.amazon.com/rds/?nc1=h_ls. [Accessed: Mar. 8, 2025].

[10] "Cloud Run," *Google Cloud*. https://cloud.google.com/run?hl=tr [Accessed: Apr. 10, 2025].

[11] Google Developers, "Custom Search JSON API introduction," Google, [Online]. Available: https://developers.google.com/custom-search/v1/introduction. [Accessed: Mar. 8, 2025].

[12] J. Thorne et al., "Automated fact-checking: Tasks, datasets, models, and challenges," arXiv preprint arXiv:2203.05659, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.05659. [Accessed: Mar. 8, 2025].

[13]"Checkmate Survey / Checkmate Anketi," *Google Docs*. https://docs.google.com/forms/d/e/1FAIpQLScg1iBTcjGqeX3B-hX12y0whNyyko6CfgKyduYR8g2Dtp_zOQ/viewform?usp=dialog

[14] Google Cloud, "Cloud Vision API," Google Cloud, [Online]. Available: https://cloud.google.com/vision?hl=tr. [Accessed: Mar. 8, 2025].

[15] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," In *EMNLP*, 2014. [Online]. Available: https://nlp.stanford.edu/projects/glove/. [Accessed: 08-Mar-2025].

[16] EFCSN, "Code of standards," EFCSN. [Online]. Available: https://efcsn.com/code-of-standards/. [Accessed: 16-Nov-2024].

[17] IBM, “What is GDPR?”, IBM. [Online]. Available: https://www.ibm.com/cloud/compliance/gdpr-eu. [Accessed: 25-Feb-2025].

[18] Wikipedia contributors, “API,” *Wikipedia*, Apr. 07, 2025. https://en.wikipedia.org/wiki/API [Accessed: 25-Apr-2024].

[19] IBM, “Natural language processing,” *What is NLP (natural language processing)?*, Apr. 17, 2025. https://www.ibm.com/think/topics/natural-language-processing. [Accessed: 25-Apr-2024].

[20] IBM, “Machine Learning,” *What is machine learning?*, Apr. 17, 2025. https://www.ibm.com/think/topics/machine-learning. [Accessed: 25-Apr-2024].

[21] Google, "URL," Google Support. [Online]. Available: https://support.google.com/google-ads/answer/14095?hl=en. [Accessed: 16-Nov-2024].

[22] Almeida, Luis B (2020) [1996]. "Multilayer perceptrons". In Fiesler, Emile; Beale, Russell (eds.). *Handbook of Neural Computation*. CRC Press. pp. C1-2.

[23] IBM, “What is retrieval-augmented generation?,” *IBM Think Topics*, 2023. [Online]. Available: https://www.ibm.com/think/topics/retrieval-augmented-generation. [Accessed: 25-Apr-2025]