

somes

Developer Documentation

Table of contents

1 Overview 1

2 Architecture 2

3 API 3

4 Frontends 4

4.1 Web 4

4.2 Mobile App 6

5 Scraper 6

5.1 Basic architecture and Functionality 6

5.2 Stenographic protocol parsing 8

5.3 Fetching, Caching and data quality 8

6 Dataprocessing and Persistence 9

6.1 Basic architecture and types 9

6.2 Services 11

6.3 AI 12

6.3.1 SomAI-API 12

6.3.2 Task scheduling 13

6.3.3 Summarizations, key points and topics 13

6.3.4 Media coverage 14

6.3.5 Political stances and the orientation quiz 14

6.4 Delegate questions and the answer mailbox 14

6.5 MCP Server 15

1 Overview

Somes (“social frames”) is a civic-tech political transparency and participation platform that processes open government data (OGD) making it accessible to everyone.

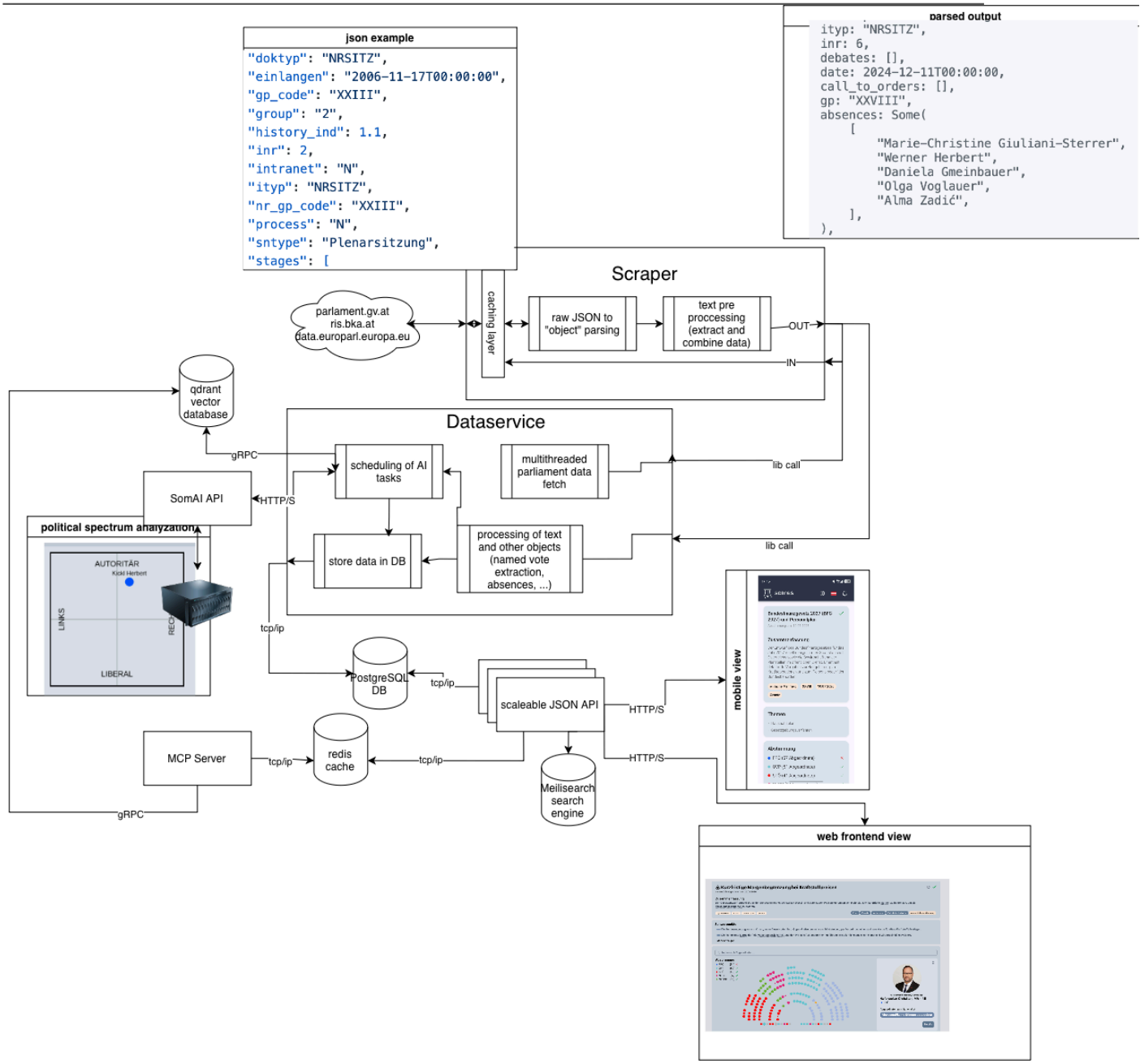
It achieves that through a diverse feature-set:

- AI enhancements including summarizations, linking to media coverage, political speech and profile analysis, ...
- data processing pipelines extracting exact roll call votes, matching attributes from raw data, combining processed data, ...
- participation approaches by allowing an exchange with members of parliaments, ...

Somes is structured via multiple repositories accessible on GitHub. The backend is primarily written in Rust, where a simple python AI REST API is the exception. The user facing applications are written in TypeScript using

Svelte (Web App) and React Native + Expo (Mobile App).
Generally, if you have any questions, please reach out to somes.austria@gmail.com.

2 ARCHITECTURE



3 API

The API related crates are located inside the somes repository. Related crates:

Crate	Purpose
somes-api	Used for serving data
somes-common-lib	Provides some api models and helper code
somes-macro	Contains procedural macros for easily turning models/types into API filterable types and composite types
somes-meilisearch-filter	Library for converting a filter type into a meilisearch filter string
views	Library for registering composite types, also contains SQL views (as the name suggests) for the main somes API models

The documentation of the API is available under https://somes.at/api/swagger-ui.

Its written in Rust using the axum-rs web framework. It handles simple CRUD operations via compile-time sqlx queries over PostgreSQL.
The (main) API process also handles filling and refreshing the meilisearch indices for members of parliament (delegates), ministerial proposals (gov_proposals), vote results, decrees and user asked delegate questions either by consuming a redis stream that is sending data changes or periodically fetching all entries. The types used to represent a single entry for a specific index also serve the purpose of custom filtering enabling queries like

https://somes.at/api/at/v1/vote_results/search?search=&page=1&is_finished=true&legislative_initiative[ityp][in][0]=I&legislative_initiative[voting][in][0]=Law&eurovoc_topics[0][topic][cn]=Wirtschaft

For this to work, every subsequent custom type of a parent type must derive MeilisearchFilter, a custom developed proc-macro implementing traits inside of the somes-meilisearch-filter library.
Multi parliament support is realised via separate maintained database and redis handles and a split inside the axum-rs Router providing the same routes for /at and /eu.

More database heavy models were abstracted into separate views using on-the-fly generated composite types used only for retrieving data.

4 Frontends

4.1 Web

The web frontend is located inside the somes repository at somes-frontend/. It is written in Svelte (SvelteKit 2, Svelte 5 runes) using TypeScript, BitsUI and Tailwind. The app is deployed via

server side rendering (SSR), nginx routes the host root to the Node process and /api to the Rust API, so every page is also rendered on the server and is indexable by search engines. Charts are built with layerchart and d3, the hemicircle seat map is plain SVG from positions calculated in lib/parliament.ts.

Relevant directories inside src/:

Directory	Purpose
routes	SvelteKit file based routing, one folder per page
lib/api	one typed function per API endpoint, additionally authed.ts for requests carrying the user token
lib/caching	client side caches for data needed on nearly every page (delegates, seats, legislative periods, party colours, plenary sessions)
lib/components	one folder per domain, e.g. VoteResults, Proposals, Decrees, Delegates, Statistics, Questions, Filtering
lib/i18n	German and English message catalogue behind one shared type
lib/persisted.svelte.ts	runes based stores that persist to localStorage
lib/seo	sitemap generation

Routes:

Route	Content
/	landing page of the association with the latest votes, drafts and decrees
/[p]/home	overview of the last plenary session
/[p]/delegates	seat map, search over all delegates, delegate profile opened through ?delegate=
/[p]/history/votes, /unfinished_votes, /proposals, /decrees	filterable and searchable lists
/[p]/vote_result/{gp}/{ityp}/{inr}, /[p]/gov_proposal/{gp}/{ityp}/{inr},	detail pages of a single vote result, ministerial proposal/draft or decree, including speeches, named votes,

/[p]/decree/{id}	media coverage and AI content
/[p]/statistics/{speech_time, absences, activity, call_to_orders, age, orientation, ...}	statistics over a parliament
/[p]/questions	delegate question system, also /questions/ask/{delegateId} /admin/delegate-questions for question moderation
/orientation	orientation quiz,
/user	/user account and notification settings

where [p] is the parliament. Multi parliament support is realised by one dynamic route segment [parliament=parliament] whose param matcher only accepts at and eu. getParliament() reads it from page.params, plink() prefixes internal links with the current parliament, and persistedScoped stores keep one localStorage slot per parliament, so filters, favourites or the AI toggle of one parliament never overwrite the ones of the other. src/hooks.ts reroutes paths that do not start with a parliament, which are all links that existed before the EU expansion, to /at. The default language is derived from the parliament too, /at renders German and /eu renders English unless the user selected a language explicitly. Because that decision happens during rendering and not only in the browser, EU pages are server rendered in English instead of flashing from German.

Filters are not mirrored in a separate state model but live directly in the URL. Every list type has a urlConversion.ts that turns its filter struct into the meilisearch query parameters described in chapter 3, and the corresponding +page.server.ts load function passes url.searchParams almost unchanged to the API, appending what the view fixes itself, for example is_finished=true on /history/votes. Every filter state is therefore bookmarkable and present in the first server response. Requests go through lib/api/api.ts, one function per endpoint, all of them returning model | HasError to match the unified error type of the API, fetchSavely converts thrown errors into that type so no call site has to catch.

Data that rarely changes, mostly the delegate list with its seat positions, is cached in localStorage and reused across pages, hot lists additionally in a CircularBuffer keyed by parliament, date and legislative period. The CacheInvalidation component refreshes party colours and plenary sessions every two minutes. The store aiViewEnabled turns every AI generated part platform wide off, summarizations, key points, titles and stance analyses alike, and AIHint marks them as AI content while the primary source stays linked.

Accounts are optional, the JWT is kept in localStorage and obtained via e-mail and password or via OAuth (google; github, discord, microsoft currently not configured) against /api/oauth/{provider}, the API redirects back to /resolve_token?token=. Only notifications, favourites, the mood barometer and the MCP token generation need an account.
For search engines /sitemap-index.xml and /sitemap/{kind}.xml serve the static pages plus decrees, ministerial proposals, questions and vote results, with lastmod taken from the API.

For development, npm run dev starts vite with host enabled so the mobile app of chapter 4.2 can point at the same server. Checks run via npm run check (svelte-check), npm run lint (eslint and prettier).

4.2 Mobile App

The documentation can be found inside the separate repository https://github.com/som-es/somes-mobile-app (README.md, /docs).

5 Scraper

5.1 Basic architecture and Functionality

The scraping of all open government data happens inside the separate repository github.com/som-es/scraper. It is the Rust library crate somes-scraper and contains no binary, no database access and no scheduling, it only turns remote documents into typed Rust structs. The dataservice repository adds it as a path dependency and writes those structs into PostgreSQL, which keeps the scraper reusable on its own for other transparency projects.

The library is split into three layers.

Layer	Purpose
common-scrapes	sub crate with the serialisable scraped types, delegates, parties, proposals, legislative initiatives, speeches, Volksbegehren and so on, re-exported at the library root
src/scraper	one module per data type, each one behind a trait, e.g. Plenar, LegislativeInitiatives, ProposalScraper, DelegateScraper, Parties, GovOfficials, RisScraper
src/text_extractor, src/text_utils	getting plain text out of the stenographic HTML protocols

Both parliaments implement the same traits, only the structs differ.

Struct	Source
AustriaScraper	parlament.gv.at, more exactly the JSON variant of its website API, every gegenstand path is requested with ?json=true and list endpoints are filled by POSTing a filter payload

EuParliamentScraper	the Open Data Portal of the European Parliament at data.europarl.europa.eu/api/v2 in JSON-LD

The Scraper trait carries BASE_URL, JSON_SUFFIX, LIST_ENTRIES and the request helpers, so that a further parliament is in principle a new struct implementing the existing traits. The abstraction is not forced everywhere, the EU module is deliberately self contained, because the POST payload and gegenstand conventions are specific to the Austrian site. Where the underlying data lines up, it produces the shared common-scrapes types, where it does not, most notably per MEP roll-call votes against the party level vote results of the Austrian National Council, it introduces EU specific types and attributes instead of a bad fit. Beyond the Open Data API it parses the seating plan, the DOCEO documents, namely roll-call votes and VOT voting records as XML, the OEIL XML export and the atom feed of the procedure list, and it keeps its own list of political group colours, because the EU API does not publish any.

Module	Data
plenar	plenary sessions with their debates and speeches, absences and order calls
legis_init	legislative initiatives with their stages, named votes, hidden votes and topics
proposals, ministrial_proposals	motions and ministerial drafts
decree	decrees over the open data API of the Federal Chancellery
enactment, dates, last_updated	dates of a law passage, modification time of an entry for incremental scraping
delegates, parties, gov_officials	delegate profiles with seats and mandates, party colours, members of the government
oral_inquiries, parliamentary_questions	question hours and parliamentary requests
volksbg	Volksbegehren from the Ministry of the Interior, including historic registration weeks recovered from the wayback archive
eu_parliament	the same for the EU, delegates, groups, procedures, plenary sessions and votes

5.2 Stenographic protocol parsing

Austrian roll call votes, absences and speech texts are the involved part. They are not part of the JSON API, the API only references the stenographic protocol as an HTML document with one anchor per statement. The TextExtractor downloads the document, cuts the range from that anchor up to the next one, removes header, footer, protocol information and time markers through regexes and fits what remains to the actual statement. That fitting step normalises soft hyphens and non-breaking spaces, and drops the academic titles in front of and behind the speaking person against a list of every title that occurs in the protocols, so that the display name of a speech can be matched against a delegate id afterwards. Statements regularly run over several documents and the same anchor occurs more than once per document, the extractor therefore keeps downloaded HTML around and picks the fitting occurrence rather than the first one. Absences are read from the same protocol documents, votes out of the stages of a legislative initiative, both as party tallies and as counts, with a hidden vote, an initiative that was declined without a recorded vote, recognised on the text of the stage.

5.3 Fetching, Caching and data quality

Requests are as polite as possible and all go through src/request.rs with the fixed user agent somes/2026.1.0, they return the typed ScraperError, a 429 becomes TooManyRequests with the Retry-After duration capped at 320 seconds. Every raw response is kept in a disk cache below caching/, the directory is settable through SCRAPER_CACHE_DIR, keys are the request urls. An entry is refetched after redo_secs, and the default config draws that value randomly between twelve and ninety days, so that a full run does not hit the API concentrated in one moment. Entries of the current legislative period skip the cache while SCRAPER_IGNORE_CACHE_XXVII is true, which keeps recent data fresh although finished periods never cause traffic again. Development on the parsers therefore normally happens against the cache.

The parliament changes its API and its documents without announcements, for that reason the repository holds the HTML and XML fixtures of both parliaments in test-data/ and around 370 unit tests, some of which request the live endpoints on purpose. The examples folder works as runnable documentation, for scraping all plenary sessions of several legislative periods, for the speech extraction and for the Volksbegehren registration weeks. One endpoint of the European Parliament sits behind an AWS WAF challenge, src/waf.rs starts a headless Chromium for it, waits for the aws-waf-token cookie and continues with plain HTTP requests afterwards.

6 Dataprocessing and Persistence

The dataservice repository is the data processing between the scraper and the API. It scrapes, matches, combines and enhances data, keeps the PostgreSQL databases of both parliaments up-to-date, and runs all AI generation. It is a cargo workspace, spreading over multiple crates.

Crate	Purpose
combx	the shared library of combined types, DB row types, API models, enhancers, delegate matching, PDF and OCR text extraction
dataservice	the long running process, one service per data type, the AI task queue,

	notifications
somai-client	typed client for the AI API, described in 6.3
update-notify	publishing changed entries into the redis streams the API consumes
delegate-question-mail	generating message ids and parsing incoming answer mails
delegate-question-mail-listener	separate binary polling the IMAP mailbox for answers
unique-topics	keeping the AI topic vocabulary stable, includes a graph UI for reviewing topic clusters
upload-political-questions	uploading the catalogues of orientation quiz and political profile analysis questions

6.1 Basic architecture and types

The most important decision of this repository is that the API models are defined here and not inside the API. somes-api and views depend on combx, therefore the pipeline and the API always speak the same types, there is no second model layer to keep in sync. A single model carries everything its representations need at once.

Derive	Produces
sqlx FromRow and Type	a database row and a PostgreSQL composite type
CompositeType from somes-macro	the SQL for creating that composite type in the database
MeilisearchFilter from somes-macro	the filterable representation described in chapter 3
optional_struct	a variant with all fields optional, used for partial updates
utoipa ToSchema and schemars JsonSchema	the OpenAPI documentation and the JSON schemas the AI has to answer against

Related data is loaded through enhancers, the Enhancer trait with the PostgresEnhancer over a connection pool, so that a vote result pulls its initiative, its named votes, its speeches and

its summaries into one nested value instead of a flat row. The index enum names the five meilisearch indices and builds their uid from the parliament, for example at-vote_results, this is the same enum the API uses for the same indices.

Matching belongs to the same layer. A scraped text name has to become a delegate id, match_delegate_name_with_delegate_id compares against the name that was valid on the date of the speech, which includes the previous name of a person, drops titles, compares single name parts and falls back to a bounded Levenshtein distance. Every result carries how it was matched and whether the match was set by hand, so a wrong assignment can be found and corrected.

Everything is database first, but only in the sense of the schema. The schema lives in 59 diesel migrations, it is applied with the diesel CLI, queries are written with the compile time checked sqlx macros and there is no ORM. Both parliaments have a database with an identical schema, an AppState carries pool, redis connection, scraper limiter, the cpu AI task channel and the parliament.

Two mechanisms keep the load of a continuous scrape reasonable. Entries are compared before they are written, either by their LastUpdated from the scraper or by a hash over the scraped value, unchanged entries cause neither an update nor an index refresh. And every insert or update of an indexed type publishes the entry into the redis stream of its index through update-notify, which is what makes the API update meilisearch within seconds instead of periodically.

6.2 Services

The dataservice binary is a single tokio process with 32 worker threads, configured through clap, --force-fill to rewrite everything, --exit-after-force-fill for a one-off run, --only-latest, --only-voteable and --log-level. It prepares the app data directory, downloads the Eurovoc thesaurus and the OCR models, connects PostgreSQL and redis, and then starts the services of the parliaments enabled by the cargo features austria and eu in parallel. Every service gets its own Shutdown listener, ctrl-c stops all of them gracefully.

A service follows the same shape. It calls the scraper through its traits, asks should_update whether the entry is new or changed, writes through combx, publishes the redis notification, sleeps and runs again. The naming is consistent, a function ending in _once backfills a whole range, run_shut and run_shut_latest_gp are the loops that keep the database current. Scrapes are wrapped in scrape_untill_success, which retries with a slowly growing pause, because interrupted requests from the parliament server are normal operation and not an error. All scrapes of one parliament share one ApiLimiter, a semaphore over PARLIAMENT_API_MAX_CONCURRENT_TASKS with a default of 24, work then spreads over legislative periods and councils through tokio::spawn.

Service	Data
delegates	profiles, mandates, seats, images, government members
parties	parties, club colours per legislative period, EU groups
plenar	plenary sessions, debates, speeches, interjections, absences, order calls

legislative_initiatives	initiatives, documents, references, votes, once per range and once for the current period
ministrial_proposals	ministerial drafts
ministrial_decrees	decrees, DECREE_PAGES pages per run
dates	plenary dates, the next session shown on the landing page
volksbg	Volksbegehren
parliamentary_qa	parliamentary requests, machine readable, behind a feature, not active (grey zone)
unique_topics	the topic catalogues per language
news_linking	media coverage, see 6.3.4
political_analysis	stance analysis, see 6.3.5
ai_task_service	the AI queue, see 6.3.2
notify	user notifications, see below

Text is taken from the HTML version of a document when one exists, otherwise from the PDF through poppler, and for documents that are only scans, through OCR. The default engine is the Rust OCR of ocrs with models downloaded on startup, a cargo feature switches the same interface to tesseract. This is what makes scanned PDFs readable for the AI.

Notifications are the part of the dataservice that reaches users directly. The delegate activity service collects what happened since the last mail per followed delegate, and new items matching the topics a user follows are sent as mail from the HTML templates in src/email through lettre, or as push notification to the mobile app through the Expo endpoint in batches of 100 and truncated to the length the push body allows. The user_notification_settings table decides per user which channels and which kinds of news are used, a user without any channel simply never appears in the recipient query.

6.3 AI

All AI generation is orchestrated from this pipeline and none of it from the API, so that a slow inference never blocks a request. Inference itself is not here either, it is in the python SomAI-API.

6.3.1 SomAI-API

The SomAI-API is a FastAPI application with JWT authentication, a single API_USER whose secret is compared against a bcrypt hash, requests from localhost may log in without it, which is how

the pipeline and the API run on the same machine as the models. It exposes deliberately few and generic endpoints.

Used Endpoint	Behaviour
/api/make_llm_request	forwards system prompt and input to an OpenAI compatible inference server, llama.cpp by default, an OpenAI and a vLLM client exist as alternatives, streams the answer, prints reasoning output, and passes a JSON schema through so the model has to answer in a valid structure
/api/model	the model currently configured, stored with every generated summary
/api/embedding/embed, /api/embedding/embed_and_cosine	embeddings with sentence-transformers on CPU, deepset-mxbai-embed-de-large-v1 by default, Qwen3-Embedding-4B for similarity comparisons
/api/embedding/reranking	cross-encoder reranking, Qwen3-Reranker-4B with a prompt describing the task as judging whether a German news article about legislation is relevant to the query

The point of this split is that prompts, output structures and retries stay in Rust. The somai-client generates a JSON schema from a Rust type with schemars, sends it with the request, and parses the answer directly into that type. Where the vocabulary of an answer has to be constrained, the pipeline patches the generated schema, for example the list of allowed topics.

6.3.2 Task scheduling

The pipeline talks to the AI only through AiTask values, a boxed async closure over the database entry, a task id, a priority and a rerunnable flag, sent over a large mpsc channel to a queue. The queue is a binary heap, tasks are ordered by priority weight and by how long they have waited, one point of age per second, so that a low priority backfill task eventually runs even while new entries keep arriving. Duplicate tasks for the same entry and task id are dropped while the process runs, otherwise a failing high priority task would occupy the queue forever, and non-rerunnable tasks stay registered. The handler keeps at most a configurable number of tasks in flight and awaits them in blocks, which is the actual reason this indirection exists, our small Build and AI server (BOSGAME M5 128GB) must not be overloaded by a full backfill. The LLM queue runs one task at a time, a second queue with sixteen concurrent tasks takes the AI work that needs no language model or GPU acceleration, which is currently only the delegate video extraction and switched off for legal reasons.

6.3.3 Summarizations, key points and topics

For legislative initiatives, ministerial drafts, decrees, speeches and parliamentary requests the pipeline sends the combined document text and gets back one structure with everything the front end shows, a speaking short title, a short and a detailed summary, the full summary with key points that reference the passage in the law or in the speech they come from, and topics classified against the topic catalogue of that language. Summaries are stored in the summaries table together with the model name the AI API reported and a version. Speech summaries additionally produce the key points that are later related to the initiatives debated in the same session, which is where the circles next to the key points of a proposal come from. The topic catalogue is the quiet half of this feature, the unique-topics crate compares all topics in use against Eurovoc and commits the resulting catalogues as JSON, so that the AI classifies into a stable, translatable set of topics instead of inventing a new ones per document.

6.3.4 Media coverage

Article URLs are crawled from the sitemaps of news sites, which keeps the selection legally clean, embedded into a qdrant collection, and matched per legislative initiative by vector search over the summary, by a per provider reranking with the cross-encoder, and by a score threshold below which nothing is stored. The remaining articles are saved with provider, score and lastmod and shown on the detail page. The service runs every three hours by default and only embeds new articles.

6.3.5 Political stances and the orientation quiz

For the direction analyses and the orientation quiz, the speeches of a delegate are embedded into a qdrant collection, then a position is generated per question and delegate with a retrieval step over those speeches, the delegate context, the questions uploaded by upload-political-questions, and a system prompt that requires a citation from the retrieved passages. The answer carries the four axis values of the political compass and the raw model output next to the parsed result. Freshness is configured per part, the embeddings are refreshed less often than the stances, and a question and delegate pair that was answered recently is skipped. The same delegate profiles are what the quiz compares a user against afterwards.

6.4 Delegate questions and the answer mailbox

Questions are written on the website and sent by the API, not by this pipeline. What belongs here is the other direction, because deputies answer by mail and have no account. delegate-question-mail generates a unique Message-ID per outgoing question and parses raw mails into sender, related message ids and body, including multipart, base64 and quoted printable bodies, and removes the quoted original question from the answer.

delegate-question-mail-listener is a second binary that polls the mailbox over IMAP, matches In-Reply-To or References against the stored outgoing Message-ID and writes the answer together with the answered status of the question in one transaction. A unique raw_message_id makes this idempotent, so a restart does not duplicate answers, failed writes are simply picked up on the next poll, and messages are never deleted from the mailbox. The whole flow, mail server and database included, is tested against GreenMail in docker with a script that copies the listener into

a temporary workspace, so the parser, the matching and the migration are covered without touching a real mailbox or the production schema.

6.5 MCP Server

The Model Context Protocol server makes the platform usable from AI tools that nobody in the project wrote, an agent in ChatGPT, in Claude or in a local client asking what a delegate said about a topic. It is the repository github.com/som-es/delegate-mcp, a Rust binary using the rmcp crate with its streamable HTTP transport, served by axum on the path /mcp and reachable through the web server at https://somes.at/mcp. It holds no data of its own, it depends on the dataservice crate with default features off and only the vectordb and runner features enabled, which gives it the qdrant client, the somai-client and the same retrieval code the stance analysis uses.

Tool	Behaviour
delegate_search	forwards to the public search endpoint of the API for the given parliament, so an agent can resolve a name into a delegate id before asking anything else
rag_query_speeches	embeds the question, searches the speech collection of that parliament filtered by delegate id with a similarity threshold of 0.87, and returns the matching passages with their score and payload as JSON text

The arguments of the retrieval tool are question, delegate id, parliament and a limit capped at 35, and its description is written as an instruction to the model, short requests covering one topic each are better than one long one, the delegate name should be left out because the id is more reliable, no reranking or further model call happens in this path, and the answers have to be cited. Reusing the internal retrieval has the consequence that an agent retrieves exactly the passages the stance analysis would have found.

Authentication runs entirely over the accounts of the platform, the server has no users. A token is generated in the user profile, a card there offers generating, regenerating and revoking it and shows the endpoint address, the API creates thirty two random bytes base64url encoded with an mcp prefix, returns it once and keeps only its SHA3-256 hash, in redis under both directions, user id to token hash and token hash to user id. The auth middleware hashes the bearer token and lets the request through only if that key exists, revoking deletes both keys and the token stops working immediately. Locally the whole layer can be switched off with --no-auth.

Rate limiting is a second middleware and shows why the order of the two matters. It buffers the JSON body to read the name of the called tool, keeps a counter per tool and identity in redis with an hour of expiry, and answers with 429 above the configured limit, fifteen RAG calls per hour by default. The identity is the user id resolved from the token rather than the token itself, which is the reason the auth middleware is added last and therefore runs first, counting per token would otherwise allow resetting the quota by generating a new one. Requests without a token stay uncounted, which is only acceptable together with --no-auth.