ITLingoCloud
System Description (2026)

Vasco Marques

September 2026

Contents

1 Purpose and Scope

ITLingoCloud is a collaborative web platform for producing, governing, and consuming rigorous specifications. It organises the work of specification into a three-level containment structure: an **organisation** groups people and shared reference material, a **workspace** holds one concrete body of specification work, and a **document** is the unit that carries a file, a type, and a lifecycle. Around that structure the platform adds a registry of the specification languages in which those

documents are written, a mechanism for producing readable deliverables from a specification and a template, and conversational assistance that turns informal description into specification text.

The platform addresses two audiences at once. Practitioners write, review, and publish specifications inside a workspace, and consume published reference material from the organisation that owns it. Language custodians evolve the specification languages themselves, authoring grammars in the browser and governing which version of a language is in force across the platform. Both audiences share one authority structure: every capability described in this document is granted by a profile held at the platform level or a role held in a specific organisation or workspace.

Three cooperating parts make up the system as a user experiences it. The first is the platform itself, reached through a public website and an authenticated portal, where organisations, workspaces, documents, and languages live. The second is the editor, ITOI, a browser-based specification editor launched from a workspace, which provides language-aware editing of specification files using the grammars the platform holds. The third is the chatbot, embedded inside a workspace page, which converses with the user and can deposit the specification it produces back into that workspace as a document. The platform is the authority in all three: it decides who may read and write what, and both the editor and the chatbot receive their working context from it.

The boundaries between the three are visible to users. Work performed in the editor does not return to the platform by itself, conversation happens beside a workspace rather than inside the editor, and a language becomes usable in the editor and known to the chatbot only once the platform has published it. The sections that follow describe each part in turn, stating capabilities as observable behaviour and naming states and roles as the platform names them.

2 Actors and Authority

Authority is held at three independent levels. A platform profile establishes what a person may do to the platform as a whole, an organisation role establishes what they may do inside one organisation, and a workspace role establishes what they may do inside one workspace. A fourth designation, language maintainership, is granted per language family and is independent of all three: holding an organisation or workspace role grants no rights over any language, and maintaining a language grants no rights in any organisation or workspace.

2.1 Platform profiles

Every registered user holds exactly one platform profile:

- *Member*: the profile every newly registered portal user receives automatically
- *Manager*: a member who additionally holds write and delete authority over the organisations and workspaces of which they are an accepted member
- *Administrator*: full authority over every record on the platform, irrespective of membership

The distinction that matters most in daily use is between the administrator and everyone else. An administrator sees and may act on every organisation, workspace, membership, document, and language, whereas a member or manager sees only the organisations and workspaces in which they hold an accepted role. Creating an organisation is not a privilege of any profile: any authenticated user may create one at any time. Suspending an organisation and reactivating it are not offered anywhere in the portal and are performed through the administrative interface.

2.2 Visitors and registered users

An unauthenticated visitor may reach the home page, the terms and conditions, the catalogue of specification languages and the detail page of any language version in force, and the list of

workspaces their owners have marked public. Inside a public workspace, a visitor may read the summary, browse and download the documents that have been published, reach the specifications section and open the workspace in the editor in read-only form, and view the workspace's conversations. No other part of the platform is reachable without signing in, and no page invites a visitor to request access to anything.

Registration asks for a name, an email address, a password, and explicit acceptance of the terms and conditions. The acceptance is recorded against the account together with the version accepted, the moment of acceptance, and the network address and the browser identification of the request. No organisation is created during registration. A new account receives the *member* profile automatically. If the address used to register matches an invitation issued earlier, the corresponding memberships are created immediately and in the *accepted* status, and the user lands in the portal already belonging to the organisations and workspaces that invited them. Otherwise the first successful sign-in directs the user to a welcome page offering to create an organisation, giving a name, an activity type, and optionally a country. The user may skip that page, in which case nothing is created and the offer is not repeated.

2.3 Organisation and workspace roles

A user holds at most one role per organisation and at most one role per workspace. Each role assignment carries a status of *pending*, *accepted*, or *rejected*, and only *accepted* assignments grant anything or make the container visible to the holder.

The three organisation roles are:

- *Organisation manager*: may edit the organisation's details, invite people, change and revoke the roles of others, create workspaces, create and edit organisation documents, configure the organisation's assistance settings, and delete the organisation
- *Document manager*: may create, edit, and delete organisation documents, and create workspaces, but may not alter the organisation or its membership
- *Organisation member*: may view the organisation and create workspaces in it, and may read and download only those organisation documents that have been published

The three workspace roles mirror them:

- *Workspace manager*: may edit the workspace's properties and progress state, invite people, change and revoke roles, create, edit, and delete workspace documents, and delete the workspace
- *Document manager*: may create, edit, and delete workspace documents, but may not alter the workspace or its membership
- *Workspace member*: may view the workspace and read and download only those workspace documents that have been published

Two propagation rules connect the levels. A user whose *organisation manager* role becomes *accepted* receives an *accepted workspace manager* role on every workspace the organisation holds at that moment, immediately and without a further invitation. The rule is not applied again afterwards: a workspace created later grants the *workspace manager* role to whoever created it and to no one else, so an organisation manager who did not create it holds no role on it until invited. In the opposite direction, a user who accepts a workspace role and has no role at all in the workspace's organisation receives an *accepted organisation member* role there, so that a workspace collaborator can always see the organisational context of the work.

Write access to the editor and the right to export a generated specification into a workspace both follow one rule: they are granted to the *workspace manager* and the workspace *document*

manager, and to no one else.

3 Organisations

An organisation is created from the portal by any authenticated user, who becomes its *organisation manager* with the *accepted* status at the moment of creation. It carries a name, an activity type chosen from information technology, marketing, consulting, finance, education, and other, an optional country, a description, and a logo. Its state is *active* or *suspended*. Only the name, the activity type, and the country may be changed from the portal, on a settings page reserved to *organisation managers*.

Membership grows only by invitation. There is no way for a user to request admission: organisations are not listed publicly, and no page anywhere offers to apply. An *organisation manager* invites by supplying an email address and one of the three organisation roles. When the address belongs to an existing account, a role assignment is created in the *pending* status, and the invitee is notified in the platform and by email. When it does not, the platform records the pending invitation, sends a message inviting the recipient to create an account, and resends that message rather than duplicating the record if the same address is invited again. Such an invitation is honoured automatically at registration and produces an *accepted* role with no acceptance step, which means that an invitee who has an account decides whether to join, whereas an invitee who does not yet have one joins by the act of registering.

An invited user answers from their notifications, accepting or rejecting the invitation there. Acceptance stamps the response time, posts a message on the organisation's history, and, when the role accepted is *organisation manager*, cascades workspace management across the organisation. Rejection records the refusal the same way. A manager is told nothing beyond the confirmation that the invitation was sent: the users page lists only members whose role is *accepted*, and no page in the portal shows outstanding invitations.

The users page itself behaves differently according to the viewer. An *organisation manager* sees every accepted member, with filters and paging, together with the controls for changing a role and removing a member. Every other accepted role sees only their own row, with the organisation's total member count but no filters and no paging, and supplying filter parameters in the address does not widen that view. A manager may change any member's role and may remove any member, including a fellow manager, subject to one invariant: the last *accepted organisation manager* can neither be demoted nor removed.

Leaving is available to every accepted member. Three outcomes are possible. A member who is the only accepted member of the organisation deletes it by leaving, unless workspaces still reference the organisation, in which case the departure is refused and the workspaces must be removed or reassigned first. A member who is the last accepted manager while other members remain is redirected to a page on which they choose a successor among the other accepted members; the promotion and the departure are then carried out together. In every other case the membership is simply removed. Deleting the organisation outright follows the same workspace restriction and is reserved to an *organisation manager*.

The organisation hub presents the organisation through a fixed set of sections reached from a sidebar. Its home summarises the organisation in grouped counts: its workspaces by progress state, its documents by type and by specification language family, its members by role, and its registered assistance engines, each count linking to the corresponding filtered list. The workspaces section lists the organisation's workspaces, restricted to those in which the viewer holds an accepted workspace role, so that a manager sees all of them through the cascade while other members see only their own. The documents section holds the organisation's own documents. The users section is described above. Two further sections, the assistance configuration described

in Section 10 and the general settings, are visible only to *organisation managers*. Counts of documents shown on the home page and the documents listed in the documents section are restricted to those in the *published* state for members who do not hold an editing role.

Organisation documents are shared into the organisation's workspaces rather than copied. Inside any workspace of the organisation, its members see the organisation's *published* documents alongside the workspace's own, and may open and download them. Editing one of those documents from inside the workspace requires an organisation editing role, not a workspace one: a *workspace manager* who is only an *organisation member* may read the shared document but not change it.

4 Workspaces

A workspace is created through a guided sequence of four steps. The first asks for a name, the owning organisation, and whether the workspace is public. The second asks for planned and actual start and end dates. The third asks for a description, and the fourth presents the entries for review before the workspace is created. Any user holding an accepted role of any kind in an organisation may create a workspace in it, and the creator receives the *workspace manager* role with the *accepted* status. A workspace references at most one organisation and the reference is optional, so deleting an organisation does not destroy its workspaces. Each workspace also carries a stable key of its own, distinct from its name, by which other parts of the system identify it.

Progress is tracked by a state independent of any document lifecycle. It takes four values, and only a *workspace manager* may change it:

- *Not started*: the initial state, from which the workspace may be started or cancelled
- *Executing*: entered by starting the workspace, which records the actual start date, and from which the workspace may be concluded or cancelled
- *Concluded*: entered by concluding the workspace, which records the actual end date
- *Cancelled*: entered from either of the first two states

The workspace home offers the first three of those transitions. Returning a concluded or cancelled workspace to *not started* is accepted by the platform but has no control on that page and is performed from the administrative interface. The progression is presented rather than enforced, because no transition examines the state a workspace is in before writing the new one. The state is in any case a statement about the work rather than an access control, and a concluded workspace remains fully usable by its members.

A workspace marked public appears on a public listing that anyone may browse and filter by name, organisation, and progress state. An anonymous visitor opening it sees a reduced version of the same hub: the summary with counts of documents by type and by specification language family but not of members, the documents that are in the *published* state, the specifications section from which the workspace can be opened read-only in the editor, and the conversations. No control that changes anything is offered. A signed-in user who holds an accepted role on a public workspace is redirected from the public view to their own, which may permit writing. Making a workspace public is a setting of the workspace itself rather than of its organisation, so one organisation may hold open and closed work side by side.

The workspace hub offers six sections. Its home carries the summary and, for a *workspace manager*, the progress transitions. The documents section lists the workspace's documents together with a second list of the published documents of its organisation. The specifications section is the launching point for the editor, described in Section 8. The chat section embeds the chatbot, described in Section 9. The users section lists the members, and the settings section, reserved to a *workspace manager*, reuses the four steps of creation to edit the workspace in place, each step

saving as it is left.

Membership works as it does for organisations, with invitations by email address and role, automatic acceptance for addresses that had no account, notification of the invitee, and the rule that the last *accepted workspace manager* may be neither demoted nor removed. Leaving behaves the same way, with one deliberate difference: a workspace is never left without members, so when its sole accepted member leaves, the workspace is deleted together with whatever it still contains, and the last manager among others must promote a successor first. One difference from the organisation hub is worth stating: on the workspace users page every accepted role sees the full member list with its filters and paging, and only the actions of inviting, changing a role, and removing are restricted to the *workspace manager*.

5 Specification Documents

A document belongs either to a workspace or to an organisation, and the platform refuses to let it belong to both. It carries a name, which is the only attribute the platform requires, and then a type, a version label, an optional content language, an optional association with a specification language, and one attached file. The version label is free text with an initial value of 1.0, which the author maintains by hand. The content language is chosen from English, Portuguese, Spanish, French, German, and other, and is declared by the author rather than detected. The association with a specification language is made in two parts, a language family and, optionally, the particular version of that family under which the content was written; naming a version without its family, or a version belonging to a different family, is refused.

The platform groups documents into a coarse format derived from the file name: word-processor documents, spreadsheets, portable documents, text, and other. An extension registered by a specification language counts as text, so specification files are treated as textual content whatever their extension. Each document also records its creator, its creation date, the size of its file, and a change history covering alterations to its name, state, version, type, and language associations.

Eight document types are available: *specification*, *specification library*, *template*, *technical document*, *dataset*, *tutorial*, *prompt template*, and *other*. Most of them classify and filter without altering what the document page does. The exception is *template*, which is the only type the platform treats differently, as Section 6 describes.

The document state is a label that tells the platform, and its members, how far a document has come and, above all, whether it is public. Five values exist, with *draft* as the initial one:

- *Draft*: the working state
- *In review*: the document has been handed to reviewers
- *Approved*: the reviewers have accepted it
- *Published*: the document is released, and the only state in which it is visible to ordinary members and, in a public workspace, to visitors
- *Archived*: the document is withdrawn from use without being deleted

The states are deliberately not an enforced workflow. On the document editing page an editor sets the state directly, and any of the five may be chosen from any other; the platform does not require a document to pass through review and approval before it is published, nor does it prevent a published document from returning to *draft*. What the state governs is visibility: a document in any state other than *published* is seen only by the roles that may edit it, and *published* alone opens it to the rest of the scope. The one further constraint concerns grounding knowledge, described below.

Who may act on a document follows the containing scope. For an organisation document, the

organisation manager and the organisation *document manager* may create, edit, and delete it, and see it in every state. For a workspace document, the *workspace manager* and the workspace *document manager* hold the same rights. Every other accepted member of the scope sees, opens, and downloads only documents in the *published* state, and the listings, the counts, and the document pages are all restricted accordingly, so that a draft is invisible to them rather than merely unopenable. Deletion is permanent and is not an archive; archiving is the separate lifecycle state.

Versioning is deliberately light. Editing a document replaces its stored file and its version label in place, and the platform keeps no prior copy: a new version of a specification is either a new version label on the same document or a separate document, at the author's discretion. The change history records which of the tracked attributes changed, the version label among them, but not the content that was replaced and not the replacement of the file itself, so a file exchanged without any other edit leaves no entry at all.

A published document may be marked as grounding knowledge, which adds its text to the pool the chatbot consults, scoped to the workspace or the organisation that owns the document. The mark may be applied only while the document is in the *published* state and only when its file is textual, either a generic text format or a format registered by a specification language. A document that leaves *published* loses the mark automatically, and its text is withdrawn from the pool at the same moment, so that content which informs assisted answers is always content the platform currently publishes. Templates may never be marked.

6 Templates and Document Generation

A template is an ordinary document whose type is *template*. Three conditions distinguish it. It must name the specification language family it applies to. Its file must be one the platform can render, which means a word-processor document, a spreadsheet, or any file whose contents are readable as text. It may never be marked as grounding knowledge, and it is excluded from the documents the editor offers to import. A template may additionally carry a pattern from which the name of the generated file is composed, which the portal discards when the document's type ceases to be *template*.

Generation is offered to anyone who may read the template. This is deliberately wider than the right to create or edit it: an ordinary member of a workspace or organisation who can open a published template can generate from it, whereas only the managers and document managers of that scope can change the template itself. The generation panel appears on the template's own page, in the workspace hub, in the organisation hub, and on the view of an organisation template surfaced inside a workspace.

To generate, the user uploads a specification written in the template's language, and the platform returns the produced file directly to the browser. The result is not stored: no document is created and nothing is retained, so generation leaves the workspace unchanged. The upload must carry an extension the target language registers. The language version used is the one currently in force for the template's family; a maintainer of that family, and an administrator, may instead direct the run at a particular version, including one still in draft. Generation is refused outright when the target language is not ready for it, which means that the version is neither in force nor in preparation, that it carries no grammar, that the grammar has not been validated or the recorded validation no longer matches it, that a version in force no longer matches the grammar recorded when it was published, or that the version declares custom services whose build is missing, failed, or out of date.

Before parsing, the platform resolves the imports the uploaded specification declares. It searches the documents of the same language family in the same workspace or organisation, whatever

their type apart from templates, and appends what it finds, so that a specification split across several documents can be generated from its entry document alone. An import that matches nothing is reported as an error naming the importing file and the line, and one that matches more than one candidate as an error naming the contested declaration and the documents that declare it. The parsed specification is then exposed to the template through a uniform structure that is the same for every language: the root element, a flat list of all elements, an index by type, an index by identifier, and uniform names for identifiers and titles. A language may additionally declare friendly aliases, which are merged in at generation time.

Rendering is lenient by design. A value the template asks for and the specification does not supply renders as blank rather than aborting the run, so an incomplete specification still produces a document. The cost is that a misspelled field name is indistinguishable from an absent value, and the authoring aids described next exist largely to compensate. A text template may also declare several named output sections, in which case the run produces each of them and delivers the set as a single archive.

The template reference documents, per language, exactly what a template may address. It records the entry type of the language, every type with its fields, and for each field whether it holds a value, a reference, or a nested element, whether it is a list, and whether it is optional. It distinguishes the types a template may iterate over from those that only ever occur inside another, and offers a ready-to-copy snippet for each, together with a downloadable starter template. It is reachable without signing in for every language the visitor may see, and it falls back to generic documentation, rather than presenting an empty inventory, when the description of the grammar cannot be produced. Alongside it the platform offers an advisory check, which analyses a pasted fragment or a stored template against the language's inventory and reports syntax errors with their positions, unknown names with the closest matching suggestion, lookups keyed on a type the language does not define, and probable field misspellings. The check never blocks generation and never stores what it was given.

7 Specification Languages

Specification languages are platform-level entities. They are not owned by an organisation or a workspace, they are registered once, and everything else refers to them. The registry is organised in two layers. A **language family** carries the identity of the language: its full name, its acronym, and the set of users who maintain it. Acronyms are unique across the platform without regard to case. A **language version** belongs to exactly one family and carries the particulars of one release: a version label, a description, the file extensions specifications in that version use, and references to the documentation, the grammar, and any external parser.

7.1 Version lifecycle and governance

A version occupies one of three states:

- *Draft*: the only state in which grammar and services files may be created, changed, or removed
- *Active*: the version in force; at most one version of a family may be active at a time
- *Deprecated*: a version that was in force and has been superseded

Every version is created as a *draft*. The only transition a user performs is publication, which moves a version from *draft* or from *deprecated* to *active* and is reserved to the platform administrator. Publication has one automatic consequence: the version of the same family that was *active* becomes *deprecated*. There is no separate action for deprecating a version and no way to set a version's state directly; attempting to write the state outside publication is refused with an explicit message.

Publication is conditional on a current, successful validation of the grammar. The platform re-runs the validation itself at publication time rather than trusting anything the browser reports, and refuses to publish a version whose grammar was never validated, whose recorded validation no longer corresponds to the current grammar text, or whose validation found errors. A version that carries custom services must additionally have a current, successful build of those services, so publication can never put into force a language whose behaviour failed to build. The identity of the publisher, the moment of publication, and a fingerprint of what was published are all recorded.

Iteration happens by creating a new draft from an existing version. The new version copies the metadata and every file of its source, leaves the source untouched, and is proposed a version label derived from the source's. Withdrawing a language altogether is done at the level of the family, which hides all of its versions at once; there is no way to archive a single version.

7.2 Maintainers and administrators

Maintainers are named on the family and are shared by all of its versions. A maintainer of a family may edit the metadata of its versions, except the name and the acronym, which only an administrator may change. They may create, rename, delete, enable, and disable its grammar and services files while the version is in *draft*, run a validation without changing the state, create a new draft version from an existing one, and export and re-import the grammar and services files as a set. Every file of a version may also be downloaded together, but the remaining kinds are replaced one at a time. They may not publish.

Reserved to the platform administrator are the creation of a language, its deletion, the assignment and removal of maintainers, and the publication or reactivation of any version. The division is deliberate: a language team prepares and validates a version, and the platform decides what comes into force, because a version in force changes the behaviour of the editor, of generation, and of the chatbot for every user of the platform.

The right to edit a language is also enforced where the change is submitted, not only by hiding controls. A request to alter a grammar or services file of a version that is not in *draft* is refused even when it is made directly, and the same applies to every mutating action of the grammar editor.

7.3 Authoring grammars and services

A version's grammar is a set of files that may import one another, edited in the browser. Exactly one of them is the entry file, and the same rule applies separately to the services files. Files are constrained to plain relative paths, must carry the extension appropriate to their kind, must be valid text, and are bounded in size. Validation first assembles the grammar by inlining the imports, rejecting import cycles, imports that resolve to nothing, and imports that leave the file set, and then checks the assembled text. Only errors make a grammar invalid; warnings and hints are reported without blocking. The result is stored with a fingerprint of the text validated and is marked out of date the moment any file of the set changes, so that a stale approval can never satisfy publication.

Custom language services let a version carry behaviour a grammar cannot express, such as how names are computed, how references are resolved across files, and what additional rules a specification must satisfy. The author supplies these as files of the version, edited in the same editor, by the same maintainers, and only while the version is in *draft*. Each change to their content rebuilds them into a single artefact, and failures are reported as diagnostics beside the grammar's own rather than blocking the save; only publication insists on a successful, current build. Every creation, change of content, and deletion of a services file is written to an audit trail

that cannot subsequently be altered or removed, whereas merely enabling or disabling one neither rebuilds the artefact nor leaves a trace there.

7.4 Reaching the editor and the chatbot

The catalogue of languages is public. A visitor and an ordinary user see the versions in force of families that have not been withdrawn; a maintainer additionally sees the versions of the families they maintain, including drafts; an administrator sees everything. The grammar and services files themselves are not public merely because the metadata is: downloading them, individually or as a set, is restricted to the administrator and the family's maintainers.

The editor obtains languages by asking the platform for them whenever it needs them, rather than receiving a deployment. For each language it receives the assembled grammar text, the registered file extensions, a fingerprint of the content, and the built services artefact. Only versions that carry a grammar are supplied, and of those the versions in force, the versions in preparation, and the superseded versions whose family has neither. Because the catalogue is read at the moment it is needed, a newly published version reaches the editor without any redeployment. The editor may also ask the platform for the specification corpus of one language family within the workspace it is serving, so that references across files resolve while editing.

The chatbot receives languages differently, as knowledge rather than as tooling. Any change to a version or to its files refreshes the language knowledge pool, and only versions in force of families that have not been withdrawn are projected into it. Each language contributes its assembled grammar as one entry, its validation files together as another, its example files together as a third, and each of its enabled specification files as an entry of its own. When a version ceases to be in force, or its family is withdrawn, its entries are removed, so that a version under preparation can never influence an assisted answer.

Organisations and workspaces may narrow the choice. An organisation may declare which languages are available to it, a workspace may narrow that set further and nominate a default, and the platform refuses a workspace selection that falls outside its organisation's or a default that is not among the workspace's available languages.

8 The Specification Editor

The editor is reached from the specifications section of a workspace. The page states, before the user commits to anything, whether the session that will open is writable or read-only, and the launch action opens the editor in a new browser tab. The same section exists on the public view of a public workspace, where it is available to visitors who have not signed in.

What the platform hands over is a short description of the session: the identity of the user, the name and identifier of the workspace, the name of the owning organisation, and a single flag stating whether the session may write. The flag is set from the user's workspace role and is true only for a *workspace manager* or a workspace *document manager*; every other role, and any user with no accepted role, receives a read-only session. A launch from the public view of a public workspace is always read-only, whoever performs it. The description is encrypted under a secret the platform shares with the editor.

Inside the editor the user may bring documents across from the platform. The editor asks the platform for the documents of the workspace and of its organisation, receiving for each the file name, the display title, and whether it comes from the workspace or the organisation, and it may then download the ones the user selects into the editing workspace. Three kinds of document are withheld: templates, documents whose extension is not among those the platform admits for import, and documents that carry no file at all. The editor may also request the specification

corpus of a language family within the same workspace, together with a report of any duplicate package names it contains, so that imports between specification files resolve during editing.

Content does not return by itself. The platform offers the editor no way to write a file back: every interface between the two carries content outward, and none accepts content inward except the delivery described in Section 9, which pushes a document the platform has just created. Work done in the editor therefore reaches the platform only when the author takes it out of the editor and uploads it as a document, and the two file sets are otherwise independent copies.

Failure is visible to the user only in the browser. The platform does not contact the editor when launching; it merely redirects, so an editor that is unavailable produces the new tab's own connection error and the platform says nothing about it. A launch naming a workspace that does not exist, and a public launch naming a workspace that is not public, both end in a not-found response. A user who holds no accepted role never reaches the launch action at all, because the specifications page itself refuses access before rendering. When the editor calls back to the platform and the call is rejected, the platform answers with a stated reason, distinguishing a missing or unusable session description, a description carrying no workspace, a workspace that does not match the one requested, a workspace that does not exist, and an absence of access to it; how those reasons are shown to the user is the editor's affair.

9 The Chatbot

Conversation happens beside the work it concerns. The chatbot is embedded in the chat section of a workspace hub and in the equivalent page of a public workspace, and nowhere else: there is no organisation-level and no platform-level conversation. A member converses in the context of one workspace, and an anonymous visitor to a public workspace reaches the same page; a signed-in member arriving at the public page is redirected to their own view of the workspace. The public page refuses a workspace that is not public. The member page does not itself check membership; what a signed-in user may do there is decided by the session statement, which is writable only for a user who in fact holds a writing role in that workspace.

The context the chatbot receives has two parts. The first is the session itself, passed when the page is rendered: the workspace's identifier and its stable key, together with a signed statement of who the user is, which organisation the workspace belongs to, and whether the session is read-only. The statement is valid for one hour and is marked read-only unless the user holds an *accepted workspace manager* or *document manager* role. The second part is the knowledge pool, which the chatbot consults rather than receives: the text of every document marked as grounding knowledge, scoped to the workspace or organisation that owns it, and the grammars, validation files, examples, and specification files of the languages currently in force.

Exporting a generated specification turns conversation output into a governed artefact. The chatbot asks the platform to create a document in the workspace, presenting the session statement as its credential and supplying the content, a proposed file name, and optionally the language the content is written in. The platform verifies the statement, checks that the workspace named matches the one the statement was issued for, and then re-derives the user's write permission from their current workspace role rather than relying on the read-only flag the statement carries. An accepted request creates an ordinary workspace document in the *draft* state, attributed to that user as its creator, with its file name made unique against the workspace's existing documents by appending a counter, so repeated exports coexist instead of overwriting one another. When a language is named, the file receives that language's registered extension whatever the conversation proposed; a language that is not currently in force is refused. The extension must in any case be one the platform admits for import, and a request that fails that test creates nothing. From that point the exported content is indistinguishable from an uploaded document: it enters the

lifecycle at *draft*, obeys the same roles and visibility, and becomes available to the editor's import on the next launch of that workspace.

Having created the document, the platform additionally attempts to deliver the same content into a live editing session, so that refinement can begin without relaunching the editor and importing the file. The two operations are deliberately not tied together. The document is the system of record and survives a failed delivery, and the platform reports the two outcomes separately: the response always names the created document and carries, beside it, whether the delivery was made, whether a live session received it, and, when it was not made, a stated reason. The reasons distinguish an integration that is not configured, a timeout, an unreachable editor, an editor lacking the delivery capability, a delivery rejected as unauthorised, another error returned by the editor, and an answer that could not be interpreted; when the editor supplies a reason of its own, that reason is passed on in place of the last but one.

A refused export is reported the same way, with the reason named rather than left to be guessed: a missing credential, an unusable or expired one, a request the platform cannot read, a missing or invalid workspace identifier, a mismatched workspace, absent content, content above one megabyte, a missing or invalid file name, an unknown or non-current language, an unknown workspace, an unknown user, an absence of write rights, and a disallowed extension are each distinguished. An export also fails, as a fault of the platform rather than of the request, when no shared secret has been configured.

When the chatbot itself cannot be reached, the platform reports nothing. The chat page renders as usual and the embedded area fails to load in the browser, because the platform never probes the chatbot before rendering. If no shared secret has been configured, the page also renders, but without any session statement, and what the user then sees is determined entirely by the chatbot.

10 Organisation-Level Assistance Configuration

Assistance is governed per organisation. Organisations differ in which providers their policies and budgets permit, in whether they may send content to a hosted service at all, and in the vocabulary and instructions appropriate to their work, so the configuration sits at the organisation level beside membership and documents rather than being fixed once for the whole platform. The configuration section appears in the organisation hub only when the conversational part of the system is present, and it is reserved to *organisation managers*: every action on it refuses a user holding any other role.

The configuration has two levels. The platform administrator maintains a platform-wide catalogue, the platform defaults, and the platform instruction block, through the administrative interface. Each organisation maintains its own catalogue on its configuration page, and the entries an organisation registers are its own: a selection is only ever validated against the organisation's own catalogue, never against another organisation's or the platform's.

A catalogue is a set of providers, each grouping one or more engines. A provider is a named family of services, whether a commercial service or one the organisation hosts itself. An engine under it carries a display label, the identifier passed to the provider, and one of three credential arrangements:

- An access key alone, for a hosted service authenticated by a key
- A service address alone, for a service the organisation hosts
- Both a key and an address, for a service that requires the two together

Credentials are handled in three ways that favour safety over convenience. They are held under names the platform composes from the organisation, the provider, and the engine, so two

organisations configuring the same provider can never collide. Changing an engine's arrangement clears the credentials belonging to the arrangement abandoned, so a stale key cannot linger behind an engine that no longer uses it. Editing an engine with the key field left empty keeps the key already stored, so a manager may change a label or an identifier without re-entering, or ever redisplaying, the secret. The values themselves are held in the platform's own storage without further encryption.

Engine roles decide what each registered engine is used for. Nine roles are assignable: a default, an engine for answering from the knowledge pool, one for generating diagrams, one for interpreting images, and one for each of the five phases of specification generation, which extract the specification from the conversation, draft the language text, check it for completeness, repair it, and finalise the answer. A manager assigns to each role one engine from the organisation's catalogue; an assignment naming anything else is cleared rather than stored. A newly created organisation begins with every role unassigned, and nothing on the platform requires them to be filled before conversation is possible.

System instructions follow a two-block arrangement. The platform block is editable only by the administrator. The organisation block is editable by the *organisation manager* from the configuration page, and a revert action discards the organisation's text entirely. Until an organisation saves a block of its own, its page presents the platform's organisation-level text as the starting point, and reverting restores that same fallback.

11 Notifications and Activity

Six events produce a notification, and all six concern membership:

- An organisation invitation is created, notifying the invited user
- An organisation invitation is accepted, notifying every accepted *organisation manager* of that organisation
- An organisation invitation is rejected, notifying the same managers
- A workspace invitation is created, notifying the invited user
- A workspace invitation is accepted, notifying every accepted *workspace manager* of that workspace
- A workspace invitation is rejected, notifying the same managers

No document, language, editor, or conversation event produces a notification. Ordinary members and document managers are not told when someone accepts or rejects an invitation; only the managers of the affected container are.

Email accompanies only the two events that create an invitation, and only when the invited user has an email address on record. The message is queued rather than sent at once, so its arrival is not simultaneous with the in-platform notification. Acceptances and rejections produce no email at all.

Users read their notifications in two places. A bell in the site header shows the number of unread notifications and the ten most recent of them, and offers to answer an invitation or mark a notification read without leaving the page. A full list page shows every notification, newest first, with paging, an action to mark all of them read, and, for an invitation still awaiting an answer, the accept and reject actions themselves. Every one of those actions is restricted to the notifications of the user performing it, and a user sees only their own notifications; the administrator sees all of them.

Beyond notifications the platform keeps an activity record. Organisations, workspaces, and documents each carry a history of the changes made to their tracked attributes, and accepting or

rejecting an invitation posts a corresponding message on the organisation or workspace concerned, so the membership history of a container can be read from the container itself. Creating an invitation also schedules a task for the invited user, which reaches them through the platform's own activity mechanism in addition to the notification.

12 Platform Administration

A small set of capabilities belongs to the platform administrator alone, and none of them is delegable through an organisation or workspace role.

Account management is reached through the administrative interface: viewing and editing every user, deleting users, and assigning the platform profile each user holds. That capability is the one item in this section the platform's own access rules do not govern. It rests on the general settings permission of the underlying account system, which the administrator profile does not itself confer, although in practice the same accounts hold both. The administrator profile is not granted automatically to anyone.

The administrator is also the only actor with unrestricted sight. Every organisation, workspace, membership, document, language version, and notification is visible and actionable to the administrator irrespective of membership, whereas every other profile sees only the containers in which it holds an *accepted* role. Suspending and reactivating an organisation are performed from the administrative interface and are not offered anywhere in the portal; deleting an organisation that one does not belong to is likewise an administrative act.

Language governance is divided as Section 7.2 describes. Creating a language, deleting one, naming and removing its maintainers, and publishing or reactivating any version are the administrator's, while the preparation and validation of a draft belong to the maintainers.

The administrator owns the platform-wide assistance settings: the catalogue of providers and engines that serves as the platform default, the default assignment of each engine role, and the platform instruction block that precedes every organisation's own. These are edited through the administrative interface rather than through any organisation's configuration page.

Finally, the administrator configures the integration with the editor, which comprises the address at which the editor is reached, the secret shared with it, the lifetime of the separately issued access credentials, and the general-purpose file extensions the platform admits for import in addition to those the registered languages contribute. The extensions of every language version in force or in preparation are admitted automatically and are not listed there, a draft's under its distinguishing suffix; a superseded version contributes its extension as well, but only for as long as its family holds no version in force and none in preparation. The setting is consequential beyond administration, because the admitted extensions determine which documents the editor is offered and which files a conversation may deposit in a workspace.