INTELLICAMPUS · PRODUCT DOCUMENTATION

BRD & Architecture Doc

Colleges and Universities · OutSystems ODC · Azure AI Search

APPLICATION
IntelliCampus

PLATFORM
OutSystems ODC

DOCUMENT VERSION
2.0

BUILT FOR
Colleges & Universities

USERS
Students & Support Team

PHASES COVERED
Phase 1 – 6

Phase 1 — Core MVP
Phase 2 — Azure Infrastructure
Phase 3 — Email Automation
Phase 4 — Architecture Refactoring
Phase 5 — ReAct Agent Loop
Phase 6 — Multi-Tenant Platform

PHASE 1

Core MVP

Ticket Management & Campus Copilot

BRD & Architecture Doc

IntelliCampus · Phase 1

APPLICATION
IntelliCampus

BUILT FOR
Colleges and Universities

USERS
Students and Support Team

ROLES
User and Admin

PLATFORM
OutSystems ODC

DOCUMENT VERSION
1.0

1. Why we are building this

Students in a college or university often have questions and problems. A student may not be able to log in, may need a document, or may not understand a rule in the handbook. Today they ask over email, on WhatsApp, or by walking to an office. Nothing is tracked, answers are slow, and the same questions come again and again.

IntelliCampus solves this in two ways. First, a student can raise a support ticket and the support team works on it until it is solved. Second, a student can ask a question to a Knowledge Assistant and get an instant answer based on the college's own document, for example the Student Handbook. This reduces the load on the support team because common questions are answered automatically.

2. Who uses the application

ROLE	WHO THEY ARE	WHAT THEY CAN DO
User	A student	Raise tickets, follow their own tickets, reply, ask the Knowledge Assistant.
Admin	Support team member	See all tickets, respond, change status, and upload the knowledge source document.

3. What is included in Phase 1

This document covers the first version of the product. The features below are in scope.

Login

BRD-01

A user can log in to the application

A returning user opens the application and is shown a login page with email and password. After logging in, the user lands on the Ticket page. Without logging in, the user cannot see any application screen.

Signup

BRD-02

A new user can create an account

A new user can click Sign up from the login page. They enter their name, email and password. After signing up, the user is taken to the Ticket page. The signup page is kept simple and does not show the main menu.

Ticket Page

BRD-03

A user sees a list of tickets

After login the user sees the Ticket page. Each row shows the Ticket ID, Title, Status, when it was last updated, who last updated it, and who raised it. The user can search the list by Ticket ID or by Title.

A student (User) sees only the tickets they raised. The support team (Admin) sees every ticket from every student. The name of the individual support agent is not shown to students; students only see that the Support Team responded.

The four ticket statuses are New, In-Progress, On-Hold and Closed.

Create Ticket

BRD-04

A user can raise a new ticket

From the Ticket page a student clicks Raise a Ticket. They enter a Title, a Description, and may attach a file of any type. On save, the system gives the ticket a unique ID, sets its status

to New, and returns the student to the Ticket page where the new ticket now appears. The support team cannot raise tickets, so the button is hidden for Admin.

Act on a Ticket

BRD-05

A user and the support team can work on a ticket

Clicking a Ticket ID opens the ticket. The top shows the ticket details: title, who raised it, when it was raised, the status, who last updated it, and when. Below that is the full conversation in order, with the student's messages on one side and the support team's replies on the other. Any attached file is shown so it can be downloaded.

At the bottom there is a reply box with a file upload. A student can add a reply and tick "Close this ticket" to close it. The support team (Admin) can add a reply, upload a file, and change the status using a dropdown. The status dropdown is editable only for Admin and is read-only for a student. On save, the reply is added and the status is updated if it was changed.

Campus Copilot

BRD-06

A user can ask questions and get answers

Campus Copilot is a simple chat page. The user types a question and receives an answer in the same screen. The user can keep asking more questions one after another. Conversations are not saved between visits; it is plain question and answer. The answer is based on the knowledge source document that the admin has uploaded, not on general internet knowledge.

Knowledge Source

BRD-07

An admin can upload the document the assistant answers from

This screen is visible to the Admin only. The admin uploads one knowledge source document, for example the Student Handbook, by entering a Source Title and a Source Name and choosing a file. The file must be a PDF and must be less than 10 MB. If the file is not a PDF or is too large, the system blocks the upload and shows a clear message.

The application keeps only one active document at a time. When the admin uploads a new document, it replaces the existing one. The Knowledge Assistant then answers all student questions using this latest document.

4. How Campus Copilot works (in simple words)

When a student asks a question, the application does not read the whole handbook every time. The uploaded PDF is first broken into small pieces. Each piece is stored in a search index in Azure AI Search. When a question comes in, the application finds the few pieces that are most related to the question, gives those pieces to the AI model, and the model writes an answer using only that information. This pattern is called RAG, which stands for Retrieval Augmented Generation.

> **In short:** Azure AI Search finds the right part of the handbook, and the AI model turns it into a clear answer. The student gets an accurate reply grounded in the college's own document.

5. Technology

- The application is built on OutSystems ODC.
- The search and retrieval part of the Agentic AI workflow uses Azure AI Search for the RAG pipeline.
- The uploaded knowledge source is a PDF, limited to 10 MB, with one active document at a time.

6. Assumptions and notes

- Login and signup in Phase 1 establish the flow. Full identity and password security is assumed to be handled by the platform's authentication, not built by hand.
- Access control matters: the Knowledge Source screen and all-ticket visibility are for Admin only and must be enforced on the server, not just hidden in the menu.
- Each ticket belongs to the student who raised it. Support agents act on it but do not own it.

7. Out of scope for Phase 1

- Multiple knowledge documents at the same time.
- Email or push notifications.
- Reports and analytics dashboards.
- Saving Knowledge Assistant chat history.

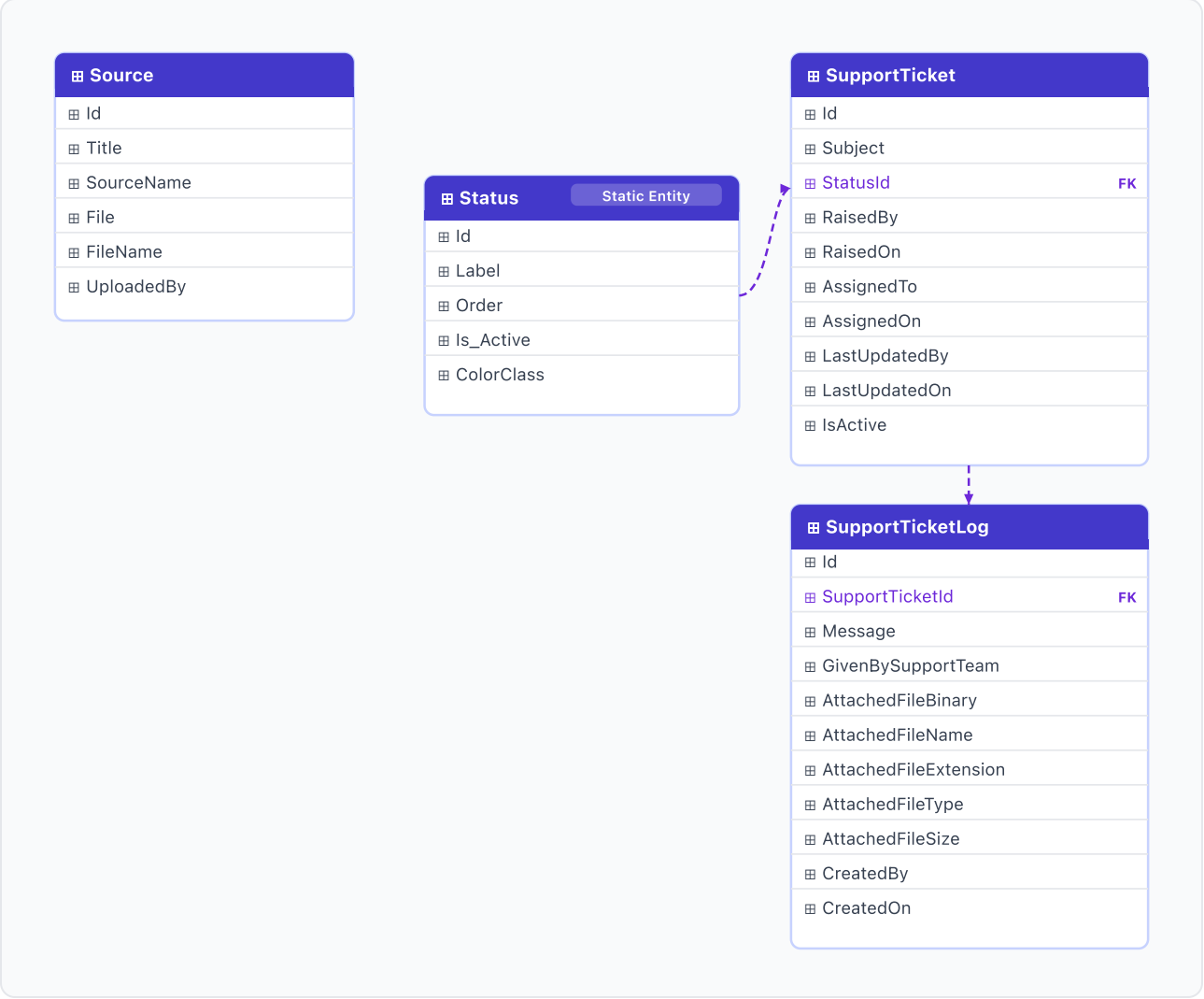
8. Architecture

Application

An OutSystems ODC Reactive Web Application named **Intelli Campus**.

Data Model

Four entities power the application. **Status** is a Static Entity — its values (New, In-Progress, On-Hold, Closed) are defined at design time, not entered by users at runtime.



ENTITY	TYPE	PURPOSE
Source	Entity	Stores the uploaded knowledge document (PDF) and its metadata.
Status	Static Entity	Defines the four ticket statuses: New, In-Progress, On-Hold, Closed. Fixed at design time.
SupportTicket	Entity	One record per support request raised by a student. References Status via StatusId.
SupportTicketLog	Entity	Every reply and file attachment in a ticket thread. References SupportTicket via SupportTicketId.

Screens

- **AddTicket** — Form for students to raise a new support ticket (title, description, optional file).
- **Copilot** — Chat interface for the AI Knowledge Assistant (Campus Copilot).
- **Signup** — New user registration. Does not show the main menu.
- **Source** — Admin-only screen to upload the knowledge source PDF.
- **Ticket** — Default screen after login; shows the ticket list with search.

- **TicketDetails** — Full ticket view with the conversation thread and reply box.

End to End Flow

1. **User** raises a ticket on AddTicket — enters a subject and description, optionally attaches a file.
2. **User** views their own tickets on the Ticket screen and opens TicketDetails to read replies.
3. **Admin** sees all tickets on the Ticket screen, opens TicketDetails, adds a reply, uploads a file if needed, and changes the status.
4. **Admin** uploads a knowledge source PDF on the Source screen. This becomes the document Campus Copilot answers from.
5. **User** asks questions on the Copilot screen and receives AI-generated answers grounded in the uploaded document.

Roles

ROLE NAME	WHO	ACCESS
`IntelliCampus_Admin`	Support team member	All tickets, Source screen, status changes, replies to any ticket.
`IntelliCampus_User`	Student	Own tickets only, AddTicket screen, Copilot. Source screen is hidden and blocked server-side.

PHASE 2

Backend Infrastructure

Self-managed Azure AI Infrastructure

BRD & Architecture Doc

IntelliCampus · Phase 2 (backend infrastructure)

APPLICATION	BUILDS ON	THEME
IntelliCampus	Phase 1	Self-managed Azure AI infrastructure

USER IMPACT	DOCUMENT VERSION
None	1.0

1. What Phase 2 changes and why

Phase 1 delivers the full product experience using a managed, shared AI infrastructure. As the institution grows, it needs full ownership of its own data, cost control, security compliance, and the ability to tune the search and AI pipeline to its specific needs.

Phase 2 is a backend-only infrastructure change. The institution provisions its own Azure Portal account, sets up Azure AI Search, defines a search index, creates an Azure OpenAI resource, and deploys an embedding model. IntelliCampus is then wired to use these self-managed resources instead of the shared ones. No student or support agent sees any difference.

Important: This is a backend change only. Campus Copilot, tickets, and all screens behave exactly as before. Only how the answer is retrieved and generated behind the scenes changes.

2. New requirements

BRD-08

Set up a dedicated Azure Portal account

The institution creates its own Azure Portal account (or uses an existing one). Under that account, a resource group is created to hold all IntelliCampus-related Azure resources.

Access to the resource group is restricted to authorised personnel only.

BRD-09

Create an Azure AI Search resource and define the search index

Within the Azure Portal, an Azure AI Search service is provisioned. A search index is defined to hold the chunked knowledge source documents. The index schema specifies the fields for the document chunk content, its vector embedding, and any metadata needed for retrieval (for example, source title and page number). The index is created before any documents are uploaded.

BRD-10

Create an Azure OpenAI resource and deploy an embedding model

An Azure OpenAI resource is provisioned under the same Azure account. An embedding model (for example, text-embedding-ada-002) is deployed within that resource. This model is used to convert both the knowledge source document chunks and incoming student questions into vector representations so that semantic similarity search can be performed in Azure AI Search.

BRD-11

Wire IntelliCampus to the self-managed Azure services

The application is reconfigured to send knowledge source documents to the institution's own Azure AI Search index (using the deployed embedding model to generate vectors) and to query that index when a student asks Campus Copilot a question. Connection strings and API keys are stored securely in the application's server-side configuration, not in front-end code.

BRD-12

No change to any user-facing behaviour

No screen, button, email, status or flow changes for students or the support team. The cut-over to the new infrastructure must not break Campus Copilot or any existing feature from Phase 1.

3. Steps to complete Phase 2

1. Create or identify the Azure Portal account and resource group.
2. Provision the Azure AI Search service and define the search index schema.

3. Provision the Azure OpenAI resource and deploy the embedding model.
4. Re-index the existing knowledge source document into the new Azure AI Search index using the new embedding model.
5. Update the IntelliCampus server-side configuration with the new endpoint URLs and API keys.
6. Test Campus Copilot end-to-end to confirm answers are retrieved from the new index.
7. Retire the old managed search dependency once the new service is confirmed live.

4. Assumptions and notes

- The existing knowledge source document must be re-indexed into the new Azure AI Search index before cut-over.
- Connection details and API keys are stored securely on the server side, never in front-end code.
- This is primarily an infrastructure and configuration effort, not a new feature for end users.
- The Azure OpenAI embedding model chosen must match the vector dimensions expected by the search index definition.

5. Out of scope for Phase 2

- Any new end-user feature.
- Supporting more than one knowledge document at a time.
- Changing the email or ticket workflows.
- Fine-tuning the AI model.

6. Architecture — Azure RAG Pipeline

Phase 2 introduces a new backend data flow for document processing. When an Admin uploads a knowledge source document, it is immediately processed through the Azure RAG pipeline so that Campus Copilot can search it semantically.

New Concept — Azure RAG Index

A new logical data store enters the architecture in Phase 2: the **Azure AI Search Index**. While it lives in Azure (not inside OutSystems ODC), it acts like a new entity from an architecture perspective. Each record in the index represents one text chunk from a Source document.

FIELD	DESCRIPTION
ChunkId	Unique identifier for the chunk.
SourceId	Links back to the Source entity in ODC so we know which document this chunk came from.

ChunkText	The raw text content of the chunk (~500 tokens).
ContentVector	The embedding vector (float array) generated by the Azure OpenAI embedding model.
PageNumber	Original page in the PDF, used for citation in answers.

Document Upload Flow (Phase 2)

1. **Upload** — Admin uploads a PDF via the Source screen. The file is saved to the Source entity in the OutSystems ODC database.
2. **Chunking** — An OutSystems Extension (built using a .NET PDF library) splits the PDF into overlapping text chunks of approximately 500 tokens each.
3. **Embedding** — Each chunk is passed to the Azure OpenAI embedding model (e.g. text-embedding-ada-002), which converts it into a numerical vector.
4. **Indexing** — The chunk text and its vector are written to the Azure AI Search index. A SourceId metadata field links each chunk back to the Source record in ODC.
5. **Query** — When a student asks Campus Copilot a question, the question is embedded (same model) and Azure AI Search returns the most semantically similar chunks. The AI model uses those chunks to compose the grounded answer.

Key point: The Azure AI Search index lives entirely in Azure — it is not stored in OutSystems ODC. OutSystems calls the Azure AI Search REST API to read from and write to it. The chunking logic runs inside an OutSystems Extension (a .NET DLL) because OutSystems itself has no built-in PDF parsing.

PHASE 3

Email Automation

Automated AI Resolution via Email Loop

BRD & Architecture Doc

IntelliCampus · Phase 3 (add-on to Phase 2)

APPLICATION
IntelliCampus

BUILDS ON
Phase 1 and Phase 2

THEME
Automated email resolution

PLATFORM
OutSystems ODC

DOCUMENT VERSION
1.0

1. What Phase 3 adds and why

In Phases 1 and 2 a student raises a ticket and waits for the support team to reply. Many tickets are common questions that the handbook already answers. Making a student wait for a person to answer something the AI could answer in seconds is slow and wastes the support team's time.

Phase 3 makes the first response automatic. The moment a student raises a ticket, the system sends an instant acknowledgement, then has the AI attempt a resolution and emails it back. The student stays in their email inbox the whole time. If the AI answer is good enough, the ticket is effectively solved with no human effort. If the student is not satisfied, one reply moves the ticket to a real support engineer.

2. New requirements

BRD-13

Instant acknowledgement email on ticket creation

As soon as a student raises a ticket, the system sends an email to that student confirming the ticket was received, with the Ticket ID. This happens immediately, before any AI processing.

BRD-14

AI attempts a resolution and emails it automatically

Creating a ticket raises a background event. The system reads the ticket title and description, asks the AI for a likely answer using the knowledge source, and emails that answer to the student as a proposed resolution. Every AI email ends with this line:

> *This is an AI assisted resolution. If you are not satisfied and want a support engineer to take care of the request, reply with "Talk to the human".*

BRD-15

User email replies are logged against the ticket automatically

Whatever the student replies to that email is read by a Google Apps Script connected to the support mailbox. The script identifies which ticket the reply belongs to (using the Ticket ID carried in the email) and logs the reply as a communication on that ticket, with no manual effort.

BRD-16

Each reply triggers a decision: human or another AI answer

Logging a reply raises another event. The system checks what the student said:

- If the reply is "Talk to the human", the ticket is assigned to a support engineer and from then on a person handles it.
- Otherwise, the system treats the reply as a follow-up question, asks the AI again, and emails the next AI resolution. This loop can continue until the student is satisfied or asks for a human.

3. The full flow

1. Student raises a ticket. Phase 1
2. System emails an instant acknowledgement with the Ticket ID.
3. Event fires. System reads the ticket, asks the AI, and emails a proposed resolution ending with the "Talk to the human" line.
4. Student replies by email.
5. Google Apps Script reads the reply and logs it against the matching Ticket ID automatically.
6. Event fires again. If the student said "Talk to the human", assign to a support engineer. Otherwise, generate and email a new AI answer, and the loop returns to step 4.

4. Assumptions and notes

- The Ticket ID must travel in the email (subject line or body) so replies can be matched back to the right ticket reliably.
- The Google Apps Script is the bridge between the support mailbox and IntelliCampus.
- When a ticket is handed to a human, the AI loop stops for that ticket.
- The AI answers only from the uploaded knowledge source, the same as Phases 1 and 2.

5. Out of scope for Phase 3

- Choosing between multiple support engineers or skills-based routing.
- Measuring how many tickets the AI closed without a human.
- Channels other than email, for example WhatsApp or SMS.

6. Development User Stories

The following user stories break down the Phase 3 requirements into discrete development tasks.

US-1 Trigger Notification Email on Ticket Creation

As a student, when I raise a support ticket, I want to immediately receive a confirmation email so that I know the system has logged my request and I have the Ticket ID.

EMAIL DETAILS

- **To:** the student's email address
- **Subject:** `Ticket ID: <Ticket ID> | <Subject / Title of the ticket>`
- **Body:** A short confirmation message acknowledging the ticket and displaying the Ticket ID clearly.

ACCEPTANCE CRITERIA

- Email is sent immediately when the ticket record is created — no manual trigger required.
- The email subject matches the format above with the correct Ticket ID and ticket title.
- If the email fails to send, the ticket is still created; the email failure must not block ticket creation.

US-2 AI Agent Response via Email on Ticket Creation

As a student, after I raise a ticket, I want to receive an AI-generated answer by email as quickly as possible.

STEPS THE SYSTEM TAKES

- Ticket creation fires an **event** (do not call the AI synchronously).
- An event handler reads the ticket title and description, and calls the AI agent with those as the query.
- The AI response is emailed to the student. The subject line must carry the Ticket ID.
- The AI response is also logged as a communication message on the ticket.

> *This is an AI generated response. If you want to talk to a human support agent, simply reply to this email with the message **"Talk to a human"** — and please do not change the subject line.*

US-3 Expose a Webhook API for Ticket Create / Update

As the system integrator, I need a secure REST API endpoint that the Gmail Apps Script can call to either create a new ticket or add a reply to an existing ticket.

REQUEST PARAMETERS

FIELD	LOCATION	MANDATORY	DESCRIPTION
`Token`	Header	Yes	Authentication secret. Reject with HTTP 401 if invalid.
`TicketId`	URL param	No	If absent, create a new ticket. If present, update existing.
`Message`	URL param	No	Body of the student's email (latest reply only).
`UserEmail`	URL param	No	Email address of the student.
`Subject`	URL param	No	Subject line of the email.
`AssignToHuman`	URL param	No	Boolean. If true, assign to Support user and skip AI response.

US-4 Create Gmail Apps Script to Bridge Email to Webhook

As the support team, I need a Google Apps Script running against the support Gmail account so that student email replies are automatically forwarded to the system.

WHAT THE SCRIPT DOES ON EACH RUN

- Searches the inbox for unread emails (limit: 1 email per run — sufficient for POC).
- Extracts sender email, subject line, and latest reply body (quoted history stripped).
- Parses the subject line to extract the Ticket ID if present.
- Checks whether the body contains "Talk to a human" and sets `AssignToHuman` accordingly.
- Calls the Webhook API via HTTP POST with all extracted values and the authentication token.
- Marks the email as read on a 2xx response; leaves unread for retry on failure.

US-5 End-to-End Testing

As the development team, before Phase 3 is declared complete, all components must work together correctly end-to-end.

TEST SCENARIOS

- **Scenario A** — Student raises a ticket from the web app → receives acknowledgement email + AI response.
- **Scenario B** — Student replies to AI email → reply appears in ticket thread → new AI response sent.
- **Scenario C** — Student replies "Talk to a human" → ticket assigned to Support → no further AI emails.
- **Scenario D** — Student emails support directly → new ticket created → AI flow triggered.
- **Scenario E** — Webhook called without Token → HTTP 401 returned.

7. Gmail Apps Script Reference

```
const API_URL = "https://<your-env>.outsystems.app/CampusIQ/rest/Webhook/Email";
const TOKEN   = "123";

function processUnreadEmails() {
  const threads = GmailApp.search("is:unread in:inbox", 0, 1);
  if (threads.length === 0) { Logger.log("No unread emails."); return; }
```

```
  const MIN_DATE = new Date(2026, 6, 12);
  const thread  = threads[0];
  const message = thread.getMessages()[thread.getMessages().length - 1];

  if (message.getDate() < MIN_DATE) { message.markRead(); return; }

  const from = message.getFrom();
  let userEmail = from;
  const emailMatch = from.match(/<(.+?)>/);
  if (emailMatch) userEmail = emailMatch[1];

  const subject     = message.getSubject();
  let   ticketId    = "";
  const ticketMatch = subject.match(/Ticket\s*ID\s*:\s*(\d+)/i);
  if (ticketMatch) ticketId = ticketMatch[1];

  let body = message.getPlainBody();
  const separators = [/\nOn .*wrote:/i, /\nFrom:/i, /\n-----Original Message----
-/i];
  for (const sep of separators) body = body.split(sep)[0];
  body = body.trim();

  const assignToHuman = /talk\s+to\s+a\s+human/i.test(body);

  let params = [];
  if (ticketId !== "") params.push("TicketId=" + encodeURIComponent(ticketId));
  params.push("Subject="       + encodeURIComponent(subject));
  params.push("Message="       + encodeURIComponent(body));
  params.push("UserEmail="     + encodeURIComponent(userEmail));
  params.push("AssignToHuman=" + encodeURIComponent(assignToHuman));

  const url = API_URL + "?" + params.join("&");
  const response = UrlFetchApp.fetch(url, {
    method: "post", muteHttpExceptions: true,
    headers: { "Token": TOKEN }
  });
  const status = response.getResponseCode();
  if (status >= 200 && status < 300) { message.markRead(); }
}
```

PHASE 4

Architecture Refactoring & Credential Management

4-App Architecture & DB-stored Azure Credentials

BRD & Architecture Doc

IntelliCampus · Phase 4 — Architecture Refactoring & Credential Management

NEW APPS	BUILDS ON	THEME
Core (shared) & Backoffice	Phase 1 through 3	Architecture & Security

PLATFORM	DOCUMENT VERSION
OutSystems ODC	1.0

1. What Phase 4 introduces and why

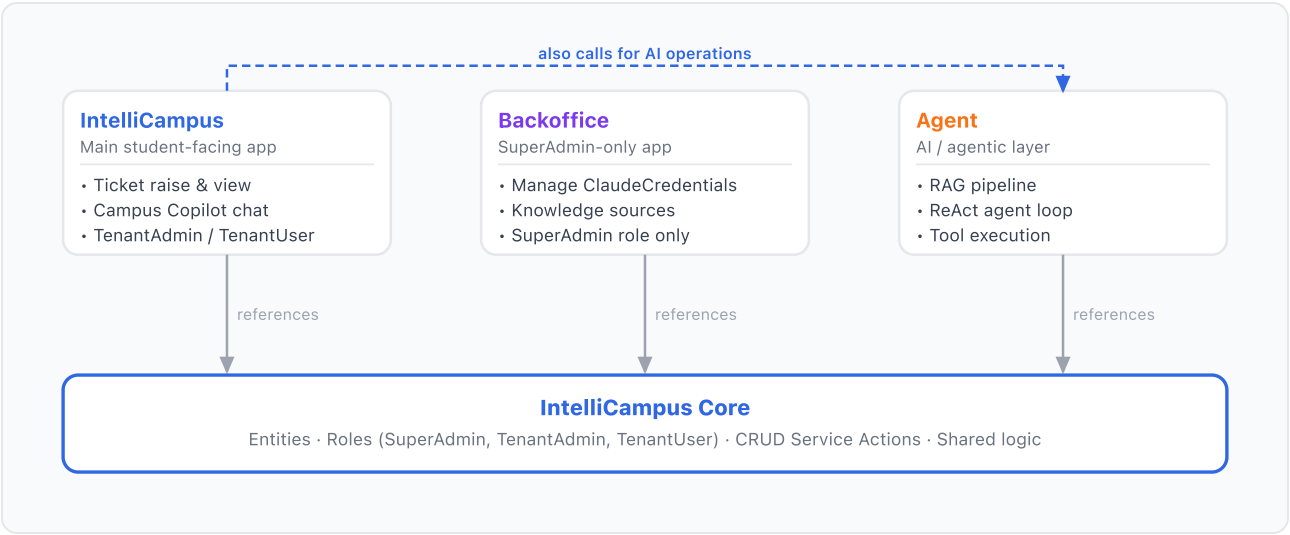
Azure credentials (endpoint URLs, admin keys, index names) are currently hardcoded across multiple files. Rotating a key requires a code deployment. At the same time, the codebase is a single monolithic application mixing data, logic, and UI.

Phase 4 fixes both problems:

- **DB-stored credentials** — Azure credentials move to a new `ClaudeCredentials` entity managed by a SuperAdmin through a dedicated Backoffice app. Key rotation requires no deployment.
- **4-app architecture** — The codebase splits into four focused ODC applications, each with a clear bounded responsibility.
- **SuperAdmin role** — A new platform-level role (`IntelliCampus_SuperAdmin`) manages system-wide configuration through the Backoffice app.

Important: No screen, button, or flow changes for students or support agents. All architectural changes are backend only.

2. Four-Application Architecture



3. Roles

ROLE	ODC NAME	APP	PERMISSIONS
SuperAdmin — New	`IntelliCampus_SuperAdmin`	Backoffice	Manage ClaudeCredentials. View all tenants. System-level configuration.
TenantAdmin	`IntelliCampus_Admin`	IntelliCampus	All tickets within tenant. Knowledge sources. User management.
TenantUser	`IntelliCampus_User`	IntelliCampus	Raise and view own tickets. Use Campus Copilot.

4. New Entity — ClaudeCredentials

Stores all Azure AI Search and Azure OpenAI credentials in the database, replacing hardcoded values. One active record at a time. Created or updated by the SuperAdmin from the Backoffice app.

FIELD	TYPE	PURPOSE
Id	Long Int (PK)	Primary key
AzureEndPointURL	Text	Azure AI Search service endpoint
AzureAdminKey	Text	Admin key for Azure AI Search (store encrypted)
AzureIndexName	Text	Name of the search index

FIELD	TYPE	PURPOSE
PerformEmbedding	Boolean	Whether to generate vector embeddings on document upload
AzureOpenAIEndPoints	Text	Azure OpenAI service endpoint
AzureOpenAIKey	Text	Azure OpenAI API key (store encrypted)
EmbeddingDeploymentName	Text	Name of the embedding model deployment
CreatedBy / CreatedOn	UserId / DateTime	Audit trail
UpdatedBy / UpdatedOn	UserId / DateTime	Audit trail

5. Migration Steps

1. Deploy the four-app split to ODC (Core, IntelliCampus, Backoffice, Agent).
2. SuperAdmin opens Backoffice and enters the existing credential values into the ClaudeCredentials form.
3. Verify Campus Copilot end-to-end to confirm the Agent is reading credentials from DB correctly.
4. Remove hardcoded credential values from all site properties and configuration files.

6. User Stories

US-P4-01 **Create IntelliCampus Core & Backoffice apps; move Source and Status entities; migrate IndexSource event**

Create two new ODC applications: **IntelliCampus Core** and **IntelliCampus Backoffice**. Move `Source` and `Status` entities to Core with full CRUD service actions. Move the `IndexSource` timer/event to Core.

KEY ACCEPTANCE CRITERIA

- Source and Status entities exist in Core, not in IntelliCampus.
- CRUD service actions exposed from Core for Source and Status.
- IntelliCampus references Core service actions instead of direct entity access.
- IndexSource timer/event exists in Core.

US-P4-02

Migrate SupportTicket and SupportTicketLog entities to Core; create service actions; migrate ProcessNewTicket event

Move `SupportTicket` and `SupportTicketLog` entities to Core. Create service actions: `CreateSupportTicket`, `GetTicketsByStudent`, `GetTicketsByTenant`, `GetTicketById`, `UpdateTicketStatus`, `AddTicketLog`, `GetTicketLogs`. Move `ProcessNewTicket` event to Core.

US-P4-03

Move roles to IntelliCampus Core; create IntelliCampus_SuperAdmin role; rebind roles in IntelliCampus screens

Remove `IntelliCampus_Admin` and `IntelliCampus_User` from IntelliCampus; recreate them in Core. Add `IntelliCampus_SuperAdmin` to Core. Rebind all role references in IntelliCampus screens and flows.

US-P4-04

Wire Backoffice app with SuperAdmin role; create ClaudeCredentials entity in Core with CRUD service actions

Configure Backoffice to reference Core and enforce `IntelliCampus_SuperAdmin`. Create `ClaudeCredentials` entity in Core. Expose `GetClaudeCredentials`, `CreateClaudeCredentials`, `UpdateClaudeCredentials` service actions.

US-P4-05

Build the Credential Management screen in IntelliCampus Backoffice

Create the primary Backoffice screen for SuperAdmin to create or update Azure credentials. On load calls `GetClaudeCredentials`. Create mode if no record exists; Update mode if record exists. Key fields render as password inputs with show/hide toggle.

US-P4-06

Use DB credentials from Core in IntelliCampus Agent and IntelliCampus apps; remove hardcoded values

Refactor IntelliCampus Agent to call `GetClaudeCredentials` before each RAG or AI operation. If no record exists, return: *"AI features are temporarily unavailable. Please contact*

support." No hardcoded Azure credentials should remain anywhere in the codebase.

PHASE 5

ReAct Agent Loop & Campus Advisor

Multi-tool AI Decision Making

BRD & Architecture Doc

IntelliCampus · Phase 5 — ReAct Agent Loop & Campus Advisor

FEATURE	BUILDS ON	PATTERN
Campus Advisor (multi-tool agent)	Phase 1 through 4	ReAct (Reason + Act loop)

PLATFORM	DOCUMENT VERSION
OutSystems ODC AI Agent	1.0

1. What Phase 5 introduces and why

The current Campus Copilot works in a single step: student asks a question → the app searches the handbook → the AI writes an answer. The AI has no decisions to make — it always does the same thing regardless of what was asked.

Phase 5 replaces this with a **Campus Advisor Agent** that reasons before acting. It looks at the question, picks the right tool, reads the result, decides if it has enough to answer — and if not, picks another tool.

- **Multi-tool decision making** — the AI chooses from four tools based on what the question actually needs.
- **Automatic ticket creation** — if no answer is found, the agent creates a support ticket on the student's behalf without asking for confirmation.
- **Multi-model architecture** — a new CampusAdvisorAgent model orchestrates tools; the existing RAGModel continues to handle document retrieval.

What changes for students: The Campus Copilot chat interface is unchanged. Behind the scenes, the AI now reasons and acts across multiple data sources before writing a response.

2. The ReAct Pattern — Plain English

ReAct stands for **Reason + Act**. Instead of doing the same retrieval every time, the AI reasons about the question, acts by calling a tool, reads the result, and decides what to do next — repeating until it is confident enough to answer.

3. Worked Example

> **Question: "When is the fee deadline and what if I miss it?"**
>
> **Think** Two sub-questions: the deadline date and the late-payment policy. Check the calendar first.
> **Act** `get_academic_calendar("fee payment deadline")`
> **Observe** "Fee deadline is October 15. Late payment incurs a 2% monthly penalty."
> **Think** Good — I have the date. Still need the waiver process from the handbook.
> **Act** `search_knowledge_base("late fee waiver appeal process")`
> **Observe** "Students may apply within 7 days by emailing finance@university.edu with documentation."
> **Think** Both answers collected. Ready to respond.
> **Answer** "The fee deadline is October 15. Missing it adds a 2% monthly penalty — but you can request a waiver within 7 days by emailing finance@university.edu."

4. Agent Tools — ODC Server Actions

TOOL NAME	INPUT	OUTPUT	IMPLEMENTATION
`search_knowledge_base`	Query: Text	Result: Text	Existing RAG pipeline — calls Azure AI Search on the tenant's handbook.
`get_academic_calendar`	Query: Text	Result: Text	New Server Action — returns structured key dates (exams, deadlines, registration windows).
`get_ticket_history`	StudentId: Text	History: Text	New Server Action — aggregate query on SupportTicket entity filtered by student.
`create_support_ticket`	Title: Text, Description: Text	TicketId: Text	Existing create-ticket Server Action — register in Agent Action Calling tab.

5. Multi-Model Architecture

MODEL	PATTERN	CALLED BY	ACTION CALLING
RAGModel (existing)	One hop: retrieve then answer	`search_knowledge_base` tool	None — single purpose
CampusAdvisorAgent (new)	ReAct loop until confident	Campus Copilot screen flow	4 registered tools (see above)

The Campus Copilot talks only to **CampusAdvisorAgent**. When the agent calls `search_knowledge_base` , that tool internally calls **RAGModel**. The agent never talks to RAGModel directly.

6. System Prompt — Key Rules

- **Rule 1:** Always use a tool before answering. Never answer from general knowledge.
- **Rule 2:** If `search_knowledge_base` and `get_academic_calendar` return nothing useful, call `create_support_ticket` immediately — **do not ask the student for permission**. Inform them of the ticket ID created.
- **Rule 3:** For multi-part questions, use tools one at a time — one tool per sub-question.
- **Rule 4:** Answers must be grounded only in tool results. Do not add context from training data.
- **Rule 5:** Keep the final response short and student-friendly. Cite the handbook section when possible.

Why Rule 2 matters: If the agent asks "shall I create a ticket?", the student's reply arrives as a new message with no tool-call context in memory — the model loses the thread and breaks. Rule 2 eliminates this by keeping ticket creation within a single agent turn.

7. Implementation Steps (ODC)

1. Create 4 stub Server Actions (hardcoded return values) — one per tool. Add clear descriptions so the model knows when to call them.
2. Open the AI Model element (CampusAdvisorAgent) → Action Calling tab → register all 4 Server Actions.
3. Create Server Action `RunCampusAdvisor(Question, StudentId)` that calls the agent and returns the final answer text.
4. Wire the Campus Copilot screen to call `RunCampusAdvisor` instead of the direct RAG pipeline.

5. Test with stub tools: verify the agent calls the right tool for each question type. Then replace stubs with real logic one at a time.

PHASE 6

Multi-Tenant Platform

Two-Application Architecture & Tenant Isolation

BRD & Architecture Doc

IntelliCampus · Phase 6 — Multi-Tenant Platform (Two-Application Architecture)

APPLICATIONS	BUILDS ON	THEME
Backoffice & IntelliCampus	Phase 1 through 5	Multi-Tenant SaaS

PLATFORM	DOCUMENT VERSION
OutSystems ODC	2.0

1. What Phase 6 introduces and why

Phases 1 through 5 built IntelliCampus for a single institution. Phase 6 evolves the product into a **multi-tenant SaaS platform** — many institutions can run independently on the same codebase, each completely isolated from the others. No tenant can ever see another tenant's tickets, users, knowledge sources, or AI search data.

Phase 6 introduces two separate applications:

- **IntelliCampus Backoffice** — a new, standalone app used only by the platform operator (Backoffice User) to create tenants and set up their users.
- **IntelliCampus** (the existing student-facing app) — gains a **User Management** section so that a Tenant Admin can manage the users of their own institution.

Two applications in Phase 6:
1. **IntelliCampus Backoffice** — platform operator creates tenants and seeds their users.
2. **IntelliCampus** — adds User Management for Tenant Admins; everything else unchanged.

2. Application 1 — IntelliCampus Backoffice (new)

BRD-22

Backoffice User can create a new Tenant

The Backoffice User fills in a form to create a new tenant. Fields: **Tenant Name**, **Tenant Code** (unique), **Status** (Active/Inactive). On save, the tenant record is created and the system is ready to receive users.

BRD-23

Backoffice User can create Users for a Tenant

From the Backoffice application, the Backoffice User can create user accounts belonging to a specific tenant. Fields: **Tenant**, **Full Name**, **Email Address**, **Role** (Tenant Admin or Tenant User). The Backoffice User must seed at least one Tenant Admin per tenant.

3. Application 2 — IntelliCampus (additions only)

BRD-24

User Management section — visible to Tenant Admin only

A new **User Management** item appears in the navigation bar for Tenant Admins only. From it, the Tenant Admin can: view all users in their tenant; create a new user; edit an existing user's name or role; deactivate a user.

BRD-25

User roles within a tenant: Tenant Admin and Tenant User

ROLE	WHAT THEY CAN DO IN INTELLICAMPUS
Tenant Admin	Everything a Support Agent can do, plus access to User Management. Can view and manage all tickets. Can upload knowledge sources.
Tenant User	Raise tickets, view own tickets, use Campus Copilot. Cannot access User Management or Knowledge Source.

BRD-26

Every user sees only their own tenant's data

Every server-side query — tickets, communications, users, knowledge sources — is filtered by Tenant ID on the server side. It is not acceptable to filter only in the UI.

BRD-27

App header shows the Tenant Name, not a generic app name

> After login, the application header must display the **Tenant Name** of the logged-in user's institution (e.g. "Norvia University"), making each institution feel like the product is their own.

4. Roles across both applications

ROLE	APPLICATION	SCOPE	RESPONSIBILITIES
Backoffice User	IntelliCampus Backoffice	Platform-wide	Create tenants; create and manage users across all tenants.
Tenant Admin	IntelliCampus	Single tenant	Manage users within their institution; handle tickets; upload knowledge sources.
Tenant User	IntelliCampus	Single tenant	Raise tickets; use Campus Copilot; view own tickets only.

5. Data flow — how a new institution gets onboarded

1. Backoffice User opens **IntelliCampus Backoffice** and creates a new Tenant (e.g. "Norvia University").
2. Backoffice User creates at least one **Tenant Admin** user for that tenant.
3. The Tenant Admin logs into **IntelliCampus**. They see "Norvia University" in the header and can access **User Management**.
4. The Tenant Admin creates **Tenant User** accounts (students) from User Management.
5. Students log in. They see only Norvia University's tickets and get AI answers grounded only in Norvia's uploaded handbook.

6. Assumptions and notes

- The Backoffice application and IntelliCampus share the same database but are deployed as separate ODC applications with separate access control.
- Tenant isolation is enforced at the server/data layer. A UI-only filter is not sufficient.
- The Tenant Name displayed in the header is read from the logged-in user's session.
- A Tenant Admin created via the Backoffice can in turn create more users via User Management.
- When a user is deactivated, their data (tickets, communications) is preserved.
- The Azure AI Search knowledge-base isolation strategy (single index with Tenant ID filter vs. one index per tenant) is to be decided during technical design based on expected tenant volume and cost.

7. Out of scope for Phase 6

- Custom logos or colour themes per tenant.
- Tenant-level billing or subscription management.
- Self-service tenant sign-up.
- Cross-tenant reporting or analytics.
- Password reset or invite-email flow.

8. Development User Stories

US-1 Backoffice — Create a Tenant

As a Backoffice User, I need to be able to create a new tenant record so that a new institution can be onboarded onto the IntelliCampus platform.

FORM FIELDS

- **Tenant Name** (mandatory) — shown in the IntelliCampus app header for all users of this tenant.
- **Tenant Code** (mandatory, unique) — short alphanumeric identifier used internally.
- **Status** — Active (default) or Inactive.

ACCEPTANCE CRITERIA

- Tenant Name and Tenant Code are mandatory; saving without them shows validation errors.
- Duplicate Tenant Code is rejected before the record is saved.
- The new tenant appears in the tenant list immediately after creation.
- Inactive tenants are visible in the list but their users cannot log in to IntelliCampus.

US-2 Backoffice — Create Users for a Tenant

As a Backoffice User, I need to be able to create user accounts for any tenant so that each institution has at least one Tenant Admin who can log in.

FORM FIELDS

- **Tenant** (mandatory) — a dropdown of all existing tenants.
- **Full Name** (mandatory).
- **Email Address** (mandatory, unique per tenant).

- **Role** (mandatory) — either Tenant Admin or Tenant User.

ACCEPTANCE CRITERIA

- All mandatory fields are validated before saving.
- The same email address can be used in two different tenants (only unique within a tenant).
- A newly created user can immediately log in to IntelliCampus and sees only their tenant's data.

US-3 IntelliCampus App — User Management for Tenant Admin

As a Tenant Admin, I need a User Management section inside the IntelliCampus app so I can add and manage my institution's users without contacting the platform Backoffice team.

USER MANAGEMENT SCREEN — FEATURES

- **User list** — shows all users who belong to the Tenant Admin's tenant.
- **Add User** — create a new user within the same tenant.
- **Edit User** — change the name or role of an existing user.
- **Deactivate / Reactivate** — toggle a user's status. Deactivated users cannot log in but their tickets remain intact.

ACCESS RULES

- The Tenant Admin can only see and manage users from their own tenant. The query must filter by Tenant ID on the server side.
- A Tenant Admin cannot deactivate themselves.
- A Tenant User who navigates directly to the User Management URL must be redirected with an "Access denied" message.

US-4 IntelliCampus App — Tenant Data Isolation

As any logged-in user, I must only ever see data that belongs to my institution.

ENTITIES THAT NEED TENANT ID ADDED

- **Ticket** — add Tenant ID as a foreign key; filter all ticket list and detail queries by session Tenant ID.
- **TicketCommunication** — inherits isolation through Ticket.
- **KnowledgeSource** — add Tenant ID; Campus Copilot retrieval must filter by Tenant ID.

- **User** — already linked by tenant; ensure User Management queries filter by Tenant ID.

> **Important:** Isolation must be enforced at the server/data layer, not just by hiding menu items in the UI.

US-5 IntelliCampus App — Show Tenant Name in Header

As a logged-in user, I want to see my institution's name in the application header so that the product feels like it belongs to my institution.

ACCEPTANCE CRITERIA

- A user belonging to "Norvia University" sees "Norvia University" in the header after login.
- A user belonging to a different tenant sees their own institution's name.
- The tenant name renders correctly for all characters including accents, spaces, and ampersands.

US-6 End-to-End Testing — Multi-Tenant Platform

As the development team, before Phase 6 is declared complete, all multi-tenancy scenarios must be verified end-to-end using at least two separate tenants.

TEST SCENARIOS

- **Scenario A** — Tenant isolation in tickets: Tenant A sees only their tickets; Tenant B cannot access Tenant A ticket IDs.
- **Scenario B** — Tenant name in header: each tenant user sees their institution's name.
- **Scenario C** — User Management scope: Tenant Admin sees only their tenant's users.
- **Scenario D** — AI answers respect tenant knowledge source: answers grounded only in the correct tenant's document.
- **Scenario E** — Backoffice creates and deactivates a user: deactivated user cannot log in; their tickets remain.

> **Testing tip:** Run all scenarios using two browsers or two incognito windows side-by-side logged in as different tenants to catch any session-bleed or caching issues.