ISOTOMA

Clear Thinking

YORK:

Swinegate Court East
Swinegate
York
YO1 8AJ

TEL:

01904 313969

https://www.isotoma.com

aws partner network

Advanced
Consulting
Partner

System Architecture

Sofia Curriculum Mapping Tool

Document revision summary

Revision	Date	Summary	Initials
1.0	May 2018	Initial version	DW
2.0	Jun 2021	Review	SR
2.1	Aug 2023	Review and Update	SR
2.2	Apr 2024	Review and Update	AP

1 Introduction

1.1 Status

This software is under continuous development, so the actual architecture is, as always, something of a moving target. This document describes current system and software architecture, and associated security analysis.

1.2 Scope of this document

This is the overall architecture document for the web application environment for the Sofia curriculum mapping tool.

The web application has a dependency on the configured authentication system for the institution – i.e. the institution's own single sign-on provider. Such a system is not within the scope of this document, although its communication with the web application is outlined here.

This document has three main parts, each with differing goals: the software application – its architecture and integration capabilities; the architecture of the system used to deliver the application; and the security of the application as a whole.

1.3 Audience

This document is intended for:

1. System implementers and maintainers
2. Operators and administrators who will be working with the system

3. Architects and designers who will be interacting with the system
4. Project managers who need to understand the work package breakdown for implementation
5. Security auditors and reviewers

1.4 Related documents

For further information on specific functionality see:

- **Sofia APIs** document describing REST APIs supporting application integration
- **Sofia Exchange Format** document describing tabular format for bulk import/export of curriculum data

2 Application

2.1 Overview

2.1.1 Purpose

The purpose of Sofia is to provide both students and staff with consistent, up-to-date and accurate views of their curriculum. Students can interact with the curriculum relevant to them, and staff can browse and make changes to current and past versions, as well as preparing future versions. Where required, users – both students and staff – can have access to multiple curriculums within their institution.

2.1.2 Curriculum versions and rollover

Each curriculum may have multiple versions, usually named after the academic year in which they commence. As a shorthand, the “upcoming” version is the most recent version in the system, usually for use in a future academic year and not available to students, and the “latest” version is the next most recent version, usually the current academic year and in use by students.

When an upcoming version is ready for student use or publication, the version and its resources are cloned to make a new upcoming version through a “rollover” process, and the former upcoming version becomes the current version. For a newly created or imported curriculum, there will be an initial staff-only phase to prepare this upcoming version for publication, during which time there will be no current version.

2.1.3 Institutions, sites, programmes and curriculums

The system provides multi-tenancy, separated at the site, i.e. hostname level. Each site is grouped into an institution. As such, “tenant” and “site” can be used interchangeably.

Multi-tenancy allows each tenant to have their own data, users, and configuration. Each site may present a different brand or visual identity (colours, fonts, logos) and navigation may be offered between sites within the same institution.

Each site may contain a hierarchy of one or more “programmes” being the usual term for a set of versions of the same curriculum, identified by pathname within the site. This hierarchy may also be used to navigate programmes by department or subject, for example. Within a programme, curriculum versions are identified by a four digit number, usually a year.

2.1.4 Curriculum items, revisions and metadata

Each curriculum takes the form of a tree or hierarchy of items, usually with a set of “year” items at the root, representing years of study within a given curriculum. Within the tree there may be “folder” items for grouping, “atom” items representing knowledge or skills, items for taught units, events, assessments and learning outcomes. Each item has its own metadata tags, connections to other items, links and resources. Additionally, curriculum events and assessments may be scheduled in terms of week numbers, days and times.

Each curriculum version has its own metadata schema describing the meaning and relationship with external standards of tags applied in that version. Each set of changes made to a curriculum version's items or metadata schema creates a new revision of that curriculum version and a full revision history is available back to its initial creation. Any two revisions may be compared to identify changes made between any two points in time.

2.1.5 Users and permission levels

Each user is linked to exactly one institution, and can only ever access sites within that institution, regardless of permission level. Sufficiently privileged users within a site can view and manage which sites users have access to and their associated permission level on that site:

- reader: can view programmes and curriculums and can leave feedback.
- student: can additionally leave feedback and add personal notes and attachments to curriculum items – uniquely, students also have defined "steps" that present them a curriculum built from study years of teaching they will receive from the latest version of the curriculum at that time.
- admin: can additionally view staff-only links/resources, metadata schema and activity (revision history).
- editor: can additionally make revisions to curriculum items, links/resources, event schedules and metadata schema, and access system tools to view feedback and manage clinical metadata.
- gatekeeper: can access additional system tools to manage institution users, site permissions, site and programme settings.

Further, a user can be marked as a "superuser" granting them gatekeeper access to all sites within their institution.

2.1.6 Key use cases

There are a number of key use cases that demonstrate some core functionality of the system, or make use of features that exercise specific parts of the architecture, or are otherwise interesting. This is not an exhaustive list:

- Student data upload: student data and programme/study/academic year "steps" are uploaded by a gatekeeper using a system tool.
- Student browse and explore: a student logs in using Single Sign On and browses their own personal curriculum built from their steps, or explores the curriculum visually, filtering on metadata tags.
- Student calendar: a student views curriculum events which are occurring at a date/time of interest and is taken to relevant resources and learning outcomes.
- Student notes: a student adds notes and/or uploads files to record against an item within the curriculum. The application ensures these remain private.
- Student revision: a student lists learning outcomes relating to external standards, knowledge, skills, taught units, placements or assessments, and marks off when they have achieved those outcomes.
- Student leaves feedback: a student can leave feedback against an item within the curriculum.

- Student feedback download: an editor can download all feedback left by students against a curriculum.
- Curriculum maintenance: minor changes are made to the latest curriculum content by editors.
- Curriculum update and rollover: major changes are made to the upcoming curriculum content by editors and a new latest version is published.
- Curriculum export/import: curriculum data including tags, scheduling and resource links can be exported, edited in a spreadsheet, imported and changes applied.
- API access: external systems can read curriculum and change data, on demand, on a schedule, or in response to webhooks triggered on update or rollover.

2.2 Software architecture

2.2.1 Authentication and account provisioning

Any access to a site, beyond viewing the login page, requires a user to be logged in. Users can authenticate themselves to a site in one of two ways: Single Sign On (SSO) or username and password. Sites within an institution may have entirely different domain names, so authentication is on a per-site basis.

It is expected that almost all users will login using SSO. The only users who will be granted access using a username and password will be:

- an account allowing gatekeeper level access for administrative purposes and in case of SSO integration failure.
- any additional specific users required by the institution, who do not have an SSO account, or require access to the system prior to SSO integration being configured for the institution – for example, during the initial staff-only phase.

Any user without a password explicitly configured in Sofia can only authenticate via SSO.

Sofia accounts may be created by a gatekeeper individually or by spreadsheet upload. Any users able to log into their institution using SSO, but not having a corresponding user account in Sofia, may have a user account created on demand based on SSO user and group data, or not be permitted to login to Sofia, depending on SSO configuration.

For users authenticating with SSO, the process is such that the user's password is submitted by the user directly to the SSO provider and is never submitted to Sofia.

2.2.2 Provisions for multi-tenancy

The system uses multi-tenancy, with all institutions using shared infrastructure. Data is stored in a single database instance, and a shared database schema. This provides a simple, flat architecture, but pushes responsibility onto the application code to ensure it partitions tenant and institution data appropriately.

Steps are taken within the application code to ensure that this filtering always happens, and that only records relating to the current tenant are retrieved. In particular, identifiers for the

institution and site must be set before querying shared resources. If, due to a bug, these are not set, the application will error rather than returning unfiltered records.

If necessary, there would be scope for altering this shared data storage to allow an institution to use a separate database instance. This would come with some considerable operational overhead – both for running the additional instance, and the additional complexity of running database schema changes across multiple instances.

Similarly, the search and cache instances are shared between all tenants. Future development work would allow each curriculum to have a customised search, allowing for domain-specific synonyms.

File storage is divided to use a separate AWS S3 “bucket” for each tenant.

2.2.3 Underlying components

Sofia is delivered as a web application. It is built using the following core components:

- Ubuntu 20.04 LTS
- Python 3.10
- Django 3.2 LTS
- Node.js 18 LTS
- PostgreSQL 12.12
- Redis 7.0
- Elasticsearch 2.3

The software is written using Python with JavaScript, TypeScript, HTML and SCSS for the user interface. Node.js is used to bundle the user interface source code into static browser-executable JavaScript, HTML and CSS files.

The user interface does much of the “heavy lifting” of handling the curriculum for display, and packaging of edits to the curriculum.

Caching and task queues are provided by Redis, and free text search is provided by Elasticsearch.

2.2.4 Use of user data

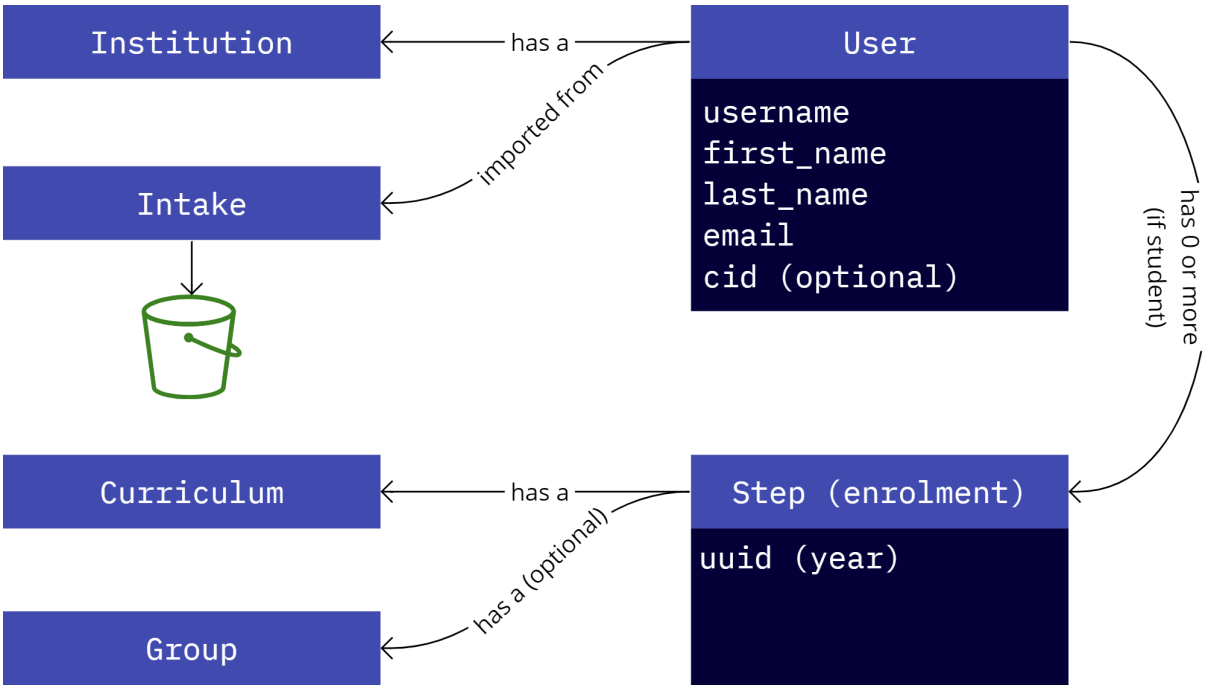
User data is stored in the database, as are student notes and feedback. This data is not stored in the Redis cache or Elasticsearch.

File attachments and user upload spreadsheets are stored in file storage. Celcat staff and student IDs are cached for 24 hours, but timetable data is not.

Any users with permission to download user feedback will receive a spreadsheet of feedback left by users, along with the details of that user (email, name and username).

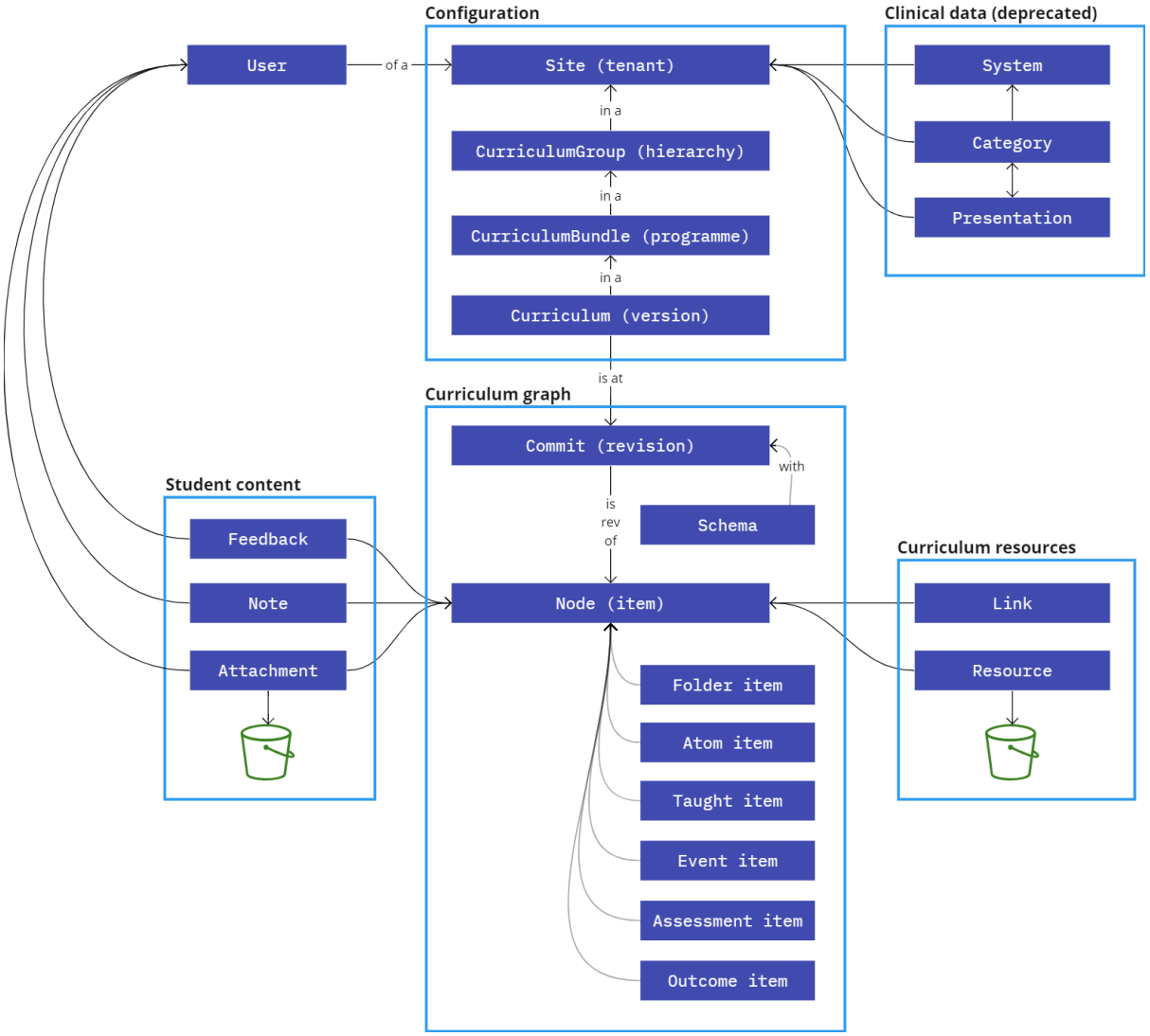
Any gatekeeper users have access to view the list of users, and are able to make changes.

The data stored for a user are given below:



2.2.5 Data model

Presented below is a schematic showing the different persistent data types handled by the application and stored in the database and in file storage:



2.3 Integrations

2.3.1 Single Sign On

The system can integrate with one SSO provider for each institution. It is expected that the number of supported SSO systems will grow to support the pre-existing SSO providers of newly added institutions. The following lists the status of some SSO systems:

- SAML 2.0: supported using redirect to IdP with POST to SP (may require customisation depending on system and configuration employed by institution)
- Active Directory and Azure AD: supported using SAML 2.0
- OAuth 2.0: expected to be supported in the future (specifically Google's OpenID Connect implementation)

At the point of login, user information in Sofia is updated from user information from the Single Sign On provider.

2.3.2 Linking and Embedding

Deep linking is supported so that user unauthenticated users accessing the URL of a specific programme/curriculum/item will be returned to that URL after authentication (whether by SSO or by username/password).

Framing is supported to allow users logged in to another system to view embedded Sofia content, authenticated by SSO or by username/password. For security reasons, the authentication process takes place in a popup window.

Framing is supported to allow users to view content from other systems embedded in Sofia, including content from systems authenticated by SSO.

2.3.3 Exchange Format

Sofia has functionality for bulk import/export of curriculum and resource link data in a human-readable tabular format, providing “round-trip” editing of data either in an external system or manually using a spreadsheet.

Import data may be used to populate an empty curriculum or to make incremental changes to an existing curriculum. Only the items which are to be changed need to be present in the import data.

The format can be created manually, by an automated process, or from a Sofia export.

For further information see the **Sofia Exchange Format** document.

2.3.4 REST API

Sofia offers REST APIs supporting integration with third-party applications. Authentication is conducted using OAuth 2.0, and offers access to endpoints including:

- Nodes - snapshot of curriculum data at a specific revision, as a list of nodes
- Tree - snapshot of curriculum data at a specific revision, as a tree of nodes
- Compare - describe changes or differences between two specific revisions
- Metadata - snapshot of metadata configuration at a specific revision and current connection configuration

Further, webhooks may be configured to inform third-party systems when new revisions and versions are created in Sofia. Webook calls are signed using a HMAC in the same way as in the Stripe API.

For further information see the **Sofia APIs** document.

2.3.5 Assessment Management Systems

The Compare and Metadata REST APIs are specifically intended for use with an AMS. These allow an AMS to periodically synchronise with any changes made to items and metadata tags and schema in Sofia.

Requests to this endpoint can ask for data relating to known revisions of a curriculum – identified by a hash – or using special values. These can be used to request changes starting from an empty curriculum (“sentinel”), or up to the most recent revision of the “latest” or “upcoming” versions of the curriculum.

Specifically, the changes endpoint returns a list of individual items, each indicating an added, modified or removed item between the two requested revisions, along with a description of the changes to each field for the item in question.

Sofia has been customised to display user interface components allowing users to interact with the Practique AMS. This customisation could be extended to support other AMS’s where appropriate.

2.3.6 Timetabling systems

Each curriculum version in Sofia may be configured to connect to one or more external timetables covering the academic year or other period the curriculum version is being delivered. Events and assessments in that curriculum may then be connected to events in the external system by ID, allowing them to be shown on Sofia on a week by week calendar, a “timeline” year planner, and a home page summary, in the same way as events using Sofia’s built in scheduling mechanism.

This approach has been used to build Celcat integration. Staff and students using Sofia see a calendar with their personal Celcat timetable, enriched with data, resources, notes and learning outcomes for each event. When browsing a curriculum staff and students see when and where they are timetabled to teach or study the events they are viewing. Staff may additionally see details of all events in the timetable relating to events in the curriculum.

This approach may be extended to support other timetabling or placement systems where appropriate.

2.3.7 Future integrations

There is potential to extend the REST API to cover other data within the system, supported by OpenAPI (Swagger) documentation, allowing access to a more comprehensive set of data types or views of that data suited to specific requirements.

The classes of systems noted below are additional candidates for integration with Sofia. The REST API may be able to provide useful integrations for some such systems, but others may require customisation, depending on the existing systems within an institution.

- Student Information System
- Virtual Learning Environment
- Curriculum Governance
- Learning Portfolio
- Communications Platform

3 System architecture

3.1 Amazon Web Services overview

The architecture used relies on various parts of Amazon Web Services to deliver a scalable and durable system. As a result of its size and widespread usage, AWS not only provides a wide range of services for delivering a system such as Sofia, but also a set of practices – both officially encouraged, and suggested by the wider community – to make best use of these services.

Where appropriate, relevant levels of service offered by AWS will be stated here without further evidence. These are widely published by AWS themselves, but should not be taken directly as an SLA regarding Sofia.

While AWS itself is a global, US-owned company, all production data stays within the “London” region. This decision was taken speculatively to greatly reduce the likelihood of future legal or compliance issues regarding location of data storage.

3.2 Principles

The following design principles apply to this architecture, and should be borne in mind for all design decisions. These are considered current best practices in the development of systems of this type.

3.2.1 Development methodology

The use of frequent, small releases as a key feature of the development and delivery approach. This maximises the value of development work, by allowing features to generate value as soon as possible. It also minimises risk, by reducing the scale and impact of each individual release.

The use of unit tests wherever possible. This reduces the amount of manual testing required, shortening release lag and enabling frequent releases.

The use of separate pull-requests per ticket, and peer code-review before changes are merged into the main branch.

3.2.2 Deployment methodology

The use of deployment automation wherever possible. This minimises the cost of creating new environments and changing existing environments, reducing friction. This also minimises the risk of human error and reduces the cost of ongoing support.

3.2.3 Data consistency

Key information is maintained within two systems for a given institution, namely the Single Sign On provider, and in Sofia. Both of these must store information sufficient to identify a user. Sofia treats the SSO provider as the source of truth for information regarding users.

3.2.4 Fault tolerance

Audit logging will be asynchronous to ensure audit requirements do not impact delivery.

Clustering will be used as appropriate to make individual service components resilient in the face of hardware or software failure.

Alerting interfaces will be provided to indicate fault situations that require manual intervention.

3.3 System overview

3.3.1 Key operations use cases

The use cases below are necessary for the ongoing support of the site, and are both the most frequently performed at present, and will need to be conducted throughout the life of the system.

3.3.1.1 Releasing a new version of the software

Supporting safe releases of new software versions is critical. Deployments are conducted as follows:

- The lead developer tags a specific revision of the code in the repository.
- Using GitHub actions, a Docker image is built and pushed to a private repository in the GitHub Container Registry
    - Specifically, by running a series of commands to install software dependencies and the custom application code
    - Once created, the image is named according to the tag it was created from.
- The created image is then manually referenced in the configuration for the stage environment and deployed using FluxCD by pushing this change to a configuration GitHub repository. The image is automatically deployed to stage using a rolling update.
- Testing can now take place on the stage environment.
- Once completed, the procedure can be repeated on the production environment using the same image that was deployed to stage, guaranteeing that the code and dependencies that were tested on stage are indeed what is promoted to production.

While this covers the manual steps taken during the release of a new version of software, the exact process of a deployment – while automated – is of relevance, and discussed in the section below dedicated to auto scaling groups.

3.3.1.2 Making infrastructure changes

Because all infrastructure is software-defined, infrastructure changes are managed entirely using AWS's CloudFormation service and Cloud Development Kit (CDK) libraries. Changes are made to the CDK code and configuration files, reviewed, and then deployed through CodePipeline invoking CloudFormation with the new configuration.

CloudFormation also uses the concept of "change sets". Before any changes are applied to the infrastructure, a change set can be created. CloudFormation creates a change set by

comparing the existing infrastructure with the new configuration, and creates the list of steps it will undertake to achieve the change. Once reviewed, these steps can then be executed.

3.4 Environments

The scope of this system comprises a number of different environments, each with a different purpose.

3.4.1 Development environment

Developers and testers use environments under their control on a local machine. Initial development and testing takes place with a limited part of the stack that allows for easy management while exhibiting largely the same behaviour as the production system.

The primary method for mimicking the production system uses Docker and Docker Compose to both run the application and a single worker, and provides the storage services (database, cache, search and file storage), HTTPS termination and test SSO providers.

3.4.1.1 Site selection

Developers can control which site they view by controlling the Host: header transmitted by their web browser. This is done by mapping tenant domains to subdomains of localhost.

3.4.1.2 Differences from production environment

This environment is straightforward to debug using standard debugging tools, and allows for a rapid development process, and provides sufficient backing services for testing almost all of the application code. There are a few ways it does differ, however:

- Threading model running the web server
- Implementation of database, cache, search and file storage
- Implementation of SSO providers

These differences must be kept in mind during development, but – with the possible exception of SSO providers – generally present a good imitation of the functionality of the production environment.

3.4.2 Environments

3.4.2.1 Stage

The “stage” (currently known as “test”) environment serves as a platform for integration testing, as well as allowing user acceptance testing. Releases are made manually using the predefined release process.

This environment:

1. Contains no real personal information
2. Can be integrated with dummy SSO providers if needed
3. Has no privileged access to any confidential systems or data

This environment is therefore considered uncontrolled from a security perspective.

3.4.2.2 Production

The “production” environment has real production data. Only authorised system administrators can have access to the data within this environment.

3.4.3 User acceptance testing

Further to the environments listed, user acceptance testing (UAT) can be conducted within specific tenants (sites/institutions) across both environments.

- Specific UAT tenants on the production environment – allowing safe user testing of production code, for example enabling an optional feature.
- Specific UAT tenants on the stage environment – allowing safe user testing of stage code, for example a newly developed feature.
- One UAT tenant per style theme on the stage environment – allowing user visibility of any changes to a specific style theme before release to production.

3.4.4 TLS certificates and domains

Certificates must be created to allow for appropriate domain names. These certificates are created and managed using AWS Certificate Manager and installed in the CloudFront delivery systems, with no access to private keys.

3.5 Data storage

How the data storage links with other parts of the system is described in more detail below. Given here are some more general operational and compliance concerns regarding data storage.

3.5.1 Storage locations

As noted above, all production data is stored in the London region, so remains within the UK. The current stage environment (known as “test”) also uses the London region, and does not contain real user data beyond that required for login for UAT purposes.

3.5.2 Encryption

Both the database and file storage are configured to encrypt data at rest – both in the primary store, any live replicas, and any backups – using keys managed by AWS.

3.5.3 Backups

Only for parts of the system where lost data would be unrecoverable are provisions for backups taken. Further, it should be noted that these are to be used in the case of a system failure, and are not to be considered an archive.

3.5.3.1 Database

This stores the vast majority of data within the system. Daily snapshots of the instance are taken using the automated snapshots from Amazon Relational Database Service (RDS), and stored for 31 days. These offer point-in-time restore.

These snapshots are stored by AWS using their S3 object storage service, which is discussed below.

3.5.3.2 File storage

The S3 buckets used for file storage use object versioning so that any items that might be inadvertently deleted are only logically deleted. The deployment role also does not have permissions to delete buckets.

S3 is – by design – an incredibly durable system, with each object replicated within a region across multiple devices in at least three availability zones. AWS describes it as being “designed to provide 99.999999999% durability of objects over a given year”.

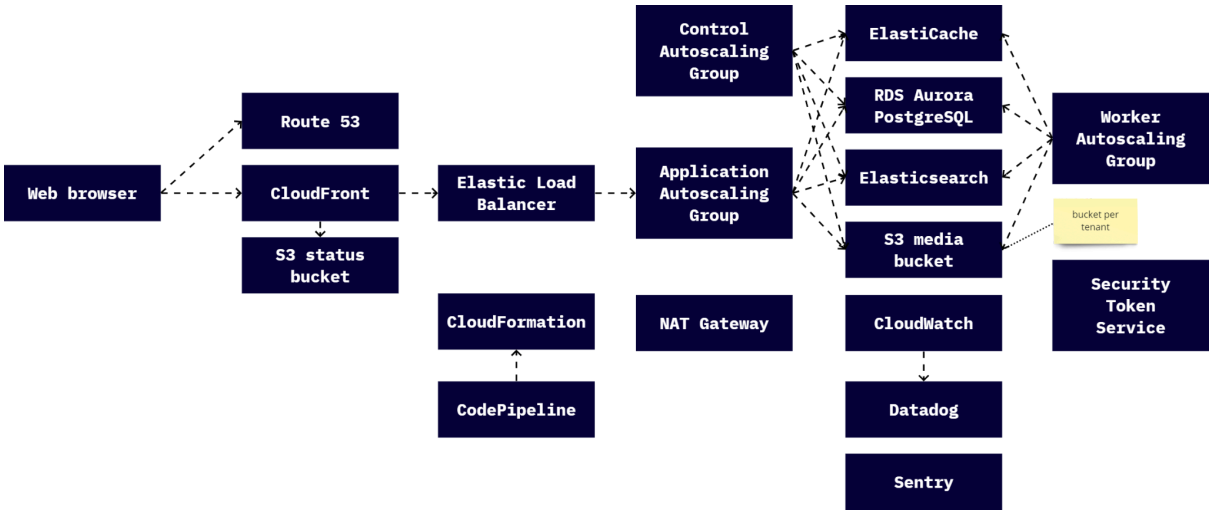
3.6 Recovery

In the event that the system suffers some catastrophic failure, a combination of a database snapshot and the required S3 buckets would be sufficient to restore the system. As the CDK code and CloudFormation templates describe the entire infrastructure, if it were necessary the whole system could be recreated.

A more plausible event requiring recovery would be that a single component of the system has failed. The architecture makes effort to make this unlikely – for example the use of a database cluster with a replica – so it would in fact require multiple replicas of a component to fail. In this case, the database could be recreated using the latest point-in-time backup available.

3.7 Amazon Web Service (AWS) configuration

3.7.1 Architecture overview



The above diagram gives a high-level overview of the components for an environment and how they link together.

3.7.2 Namespaces

The environments are partitioned from each other by being created in separate Kubernetes namespaces.

The following accounts exist and are used as noted below:

Namespace	**Purpose**
production	Production
test	Stage

3.7.3 High Availability

Refer to the diagram overleaf for a component level view of the architecture.

All services are configured in their High Availability configuration, with the exceptions of the cache and the search. How this works in practice varies from service to service, although the core concept is the partitioning of AWS physical data centres into Availability Zones (AZs).

Component	**Analysis**	**Provision for HA**
CloudFront	Key service	Designed by AWS to provide at least 99.9% uptime during any month, and implemented as a very large network of edge servers.
Elastic Load Balancer	Key service	Has a “leg” in each of the two application subnets, each in a different AZ.
Kubernetes cluster servers	Key service	Always at least 2 instances, balanced between AZs.
Application containers	Key service	Always at least 2 containers, balanced between cluster servers
Worker containers	Key service, used asynchronously	Always at least 1 container; recreated should it become unavailable.
NAT Gateway	Key service	One gateway in each AZ.
RDS instance	Key service	Has a primary and a replica in different AZs, with failover managed automatically by RDS. Such multi-AZ instances are designed by AWS to achieve at least 99.95% uptime during any month.
S3	Key service	By design, a highly available service, using eventual consistency amongst a large number of nodes.
ElastiCache	Temporary loss tolerated	Will be reprovisioned and repopulated automatically should it suffer a failure.

Elasticsearch	Temporary loss tolerated	Will be reprovisioned and repopulated automatically should it suffer a failure.

3.7.4 Identity and Access Management (IAM)

3.7.4.1 Key pairs and Server Access

No conventional SSH access is possible to the cluster instances and no key pairs are deployed. Access is instead via AWS's Session Manager, meaning no instances have an exposed SSH port.

3.7.4.2 Roles and Container Profiles

The following roles are created within IAM:

Role	**Access Summary**
ApplicationRole, per environment	Writing CloudWatch metrics and logs Elasticsearch full access S3 access for managing media
Deployment	Gives deployment tool full access
CloudTrailRole	Allows CloudTrail to push logs
DatadogAWSIntegrationRole	Read access to all services, limited to actions relevant for monitoring

For the application role, an IAM role is created for each environment and only the containers for this environment can use this role, granting access to the relevant resources for that environment.

3.7.5 Virtual Private Cloud (VPC)

3.7.5.1 Allocation of ranges to regions

The address range for each environment is 10.0.0.0/20. This is represented by a VPC within each account using this address range.

3.7.5.2 Subnets

Within the VPC, 7 subnets are created, 2 in each of two availability zones, plus two in just the primary availability zone as follows:

Name	**AZ A Subnet**	**AZ B Subnet**
application (also used for workers)	10.0.1.0/25	10.0.2.0/25
load balancer	10.0.4.0/25	10.0.5.0/25
nat	10.0.6.0/25	10.0.7.0/25

Having these subnets in two availability zones allows us to implement the high availability features offered by AWS.

3.7.5.3 Routing

All subnets can route internally within the VPC (this is a default part of the AWS configuration that cannot be changed). However subnets cannot route outside the VPC without having a specific routing table entry.

There is a single egress point from the VPC, an internet gateway, connecting the VPC to the Internet via the AWS firewalls. Egress is all via the NAT gateways.

Note that no external routes exist to either of the application subnets. The only access to these subnets is via other subnets. The load balancer, control and NAT subnets are all public.

3.7.6 Simple Storage Service (S3)

Within each environment an S3 bucket is created for each site. This bucket is used to store media (any files uploaded by users, and any generated spreadsheets). These buckets are all set to be private. They also have versioning enabled – this is used to ensure that any files that are deleted are only logically deleted and can therefore be recovered if needed.

There is one public bucket, which is used to display the fallback error page should the site ever be inaccessible.

3.7.7 Relational Database Service (RDS)

A database subnet group is created spanning both application subnets.

A single Aurora PostgreSQL cluster is created within this subnet group, and within this a database (named “curricle”) is created, with separate writer and reader instances to distribute workload.

This database has HA enabled and should transparently fail over between primary and replica instances. Further, the primary and the replica are each accessed through a separate subnet within the subnet group – specifically, the primary is within the primary application subnet and the replica is within the backup application subnet.

The data the instance stores is encrypted at rest, using keys managed by AWS. Further, 31 days of daily snapshots are retained, and each of these is also encrypted.

3.7.8 ElastiCache

A cache subnet group is created spanning both application subnets. Currently, there is only one cache instance created, made available to both subnets.

3.7.9 Amazon Machine Images

All EC2 servers are created from an AMI that is provided by AWS for use as Kubernetes servers with AWS's Elastic Kubernetes Service (EKS). The application containers are deployed onto these servers and run with Docker.

3.7.10 Elastic Load Balancers

A single load balancer 'balancer' is created in a High Availability configuration. This provides HTTP access to nodes within the application cluster only.

When containers or servers in the application cluster are terminated, connection draining is handled by the load balancer.

3.7.11 Auto Scaling Groups and Launch Configurations

All EC2 servers are created and managed by auto scaling groups and launch configurations. These allow AWS to transparently and automatically scale, rotate and manage instances between designated availability zones. One ASG of Kubernetes servers is created in each of the two AZs.

3.7.12 CloudFront

CloudFront is used to cache static assets, and to provide dedicated high-speed backhaul to the application origin containers.

There is a separate CloudFront distribution per site, with an TLS certificate installed into CloudFront for the public URL of the site.

Each distribution has two origins. A default origin that maps to the load balancer, and a /status origin that maps to the status bucket.

Further, the default origin has different behaviour for /static that enables caching for static assets from the application (e.g. JavaScript and CSS assets).

The CloudFront distribution obeys caching headers provided by the application. Middleware within Django explicitly turns off caching for all dynamic pages.

This ensures private data is never leaked from the cache, while still allowing acceleration of static content.

3.7.13 Route 53

Only the internal sofiasrv.net domain is hosted on Route 53 as part of this design. Other DNS entries, for public parts of the service, may be hosted anywhere.

Records are created in Route 53 as follows:

Record	**Environment**
sofiasrv.net	production

prod.sofiasrv.net	production
pre.sofiasrv.net	pre-production (stage)

The sofiasrv.net domain itself has been registered with GANDI and the WHOIS nameserver and root server entries point to the designated AWS nameservers.

Each production tenant is given a subdomain of prod.sofiasrv.net, and this can be used as a target for the public domain.

3.7.14 Architecture Management

The configuration for all of the above components is managed using CDK and CloudFormation. This allows multiple environments to be provisioned easily, and with confidence that they are configured identically, safe for explicit changes between environments.

3.8 Software execution environment

The backend application code itself executes in Docker containers running either “application” or “worker” commands. On these containers, the execution environment for the software comprises:

1. Ubuntu LTS base provided and patched by Ubuntu
2. Other Ubuntu packages needed for operation
3. Packages necessary to set up the Python Virtual Environment
4. Open Source Python packages produced by third parties

3.8.1 Application software

The software and its supporting infrastructure, build, installation and deployment scripts are all kept within git repositories at GitHub.

Before release to production environments this software is tagged within git. It is then built into a Docker image using GitHub Actions and pushed to GHCR.

3.8.2 JavaScript and CSS bundling

The frontend application source code is bundled into browser-executable static JavaScript and CSS files using Webpack 5 during Docker image building. These files are initially served from the container filesystem by Django and then cached by CloudFront.

3.8.3 The Python Virtual Environment

The Python Virtual Environment is constructed using three Ubuntu packages:

- python3.10
- python3.10-dev
- python3.10-venv

The virtual environment provides a constrained environment where dependencies for Python software may only be found from within this virtual environment. This ensures only Python packages intentionally installed into the virtual environment are used.

The virtual environment is populated with packages using the requirements.txt file, which is contained within the repository.

3.8.4 Docker image construction

Docker images are built using an Ubuntu LTS release. These images are automatically patched by Ubuntu. Regular rebuilding of Docker images and redeploying tasks therefore ensures that all recommended security patches are applied.

3.9 Availability

3.9.1 Target

The architecture itself has been designed with the goal of maximising availability, with redundancy of any components critical for users and staff to access the system.

However, further provisions have been made to understand the resilience, and to monitor the ongoing performance of the system.

3.9.2 Operational dependencies

The SSO provider in use for an institution will be used by almost all users to gain access to the system. As such, Sofia will be inaccessible to users should the SSO provider be unavailable. Users that authenticate with username and password will still be able to log in.

Timetable data is retrieved on demand from Celcat with limited caching in the Sofia application. As such, timetable data will not be presented should Celcat be unavailable. The rest of the application will continue to operate and will periodically retry the connection to Celcat.

3.9.3 Monitoring

Wide ranging metrics are recorded from all components of the system to allow systems administrators vision over the health and performance of the system.

These are recorded by all parts of the AWS infrastructure using CloudWatch and stored in Datadog. Additional measurements not supported by CloudWatch are made using a Datadog agent running on each cluster instance. Datadog retains all of these metrics for 15 months.

Datadog is also used to provide a dashboard for each environment, displaying key metrics.

3.9.4 Logging

Logs are shipped to Datadog using a Datadog agent running on each cluster instance, and retained for 31 days, providing more powerful log analysis for any urgent investigation.

An augmented form of logging is used to handle any errors within server-side application code. Sentry's client captures relevant data regarding the error – appropriately cleaning sensitive data such as passwords – and records this in Sentry, retained for 90 days.

3.9.5 Alerting

Alerts are generated both from monitoring, should particular metrics fall outside specified ranges, and from any errors reported to Sentry.

3.9.6 Scaling

As noted throughout the description of the architecture, the cluster instances make use of auto scaling groups, serving the dual purpose of ensuring instances are replaced should they become unhealthy, and allowing the system to increase its capacity when under load. Kubernetes handles scaling of containers based on load, and triggers scale up of the instances if more capacity is needed.

As the infrastructure is shared between tenants, the exact number of instances required will vary depending on the number of tenants, and the activity of users. Scaling will adapt the production environment to accommodate load increases.

3.9.7 Load

Some tests to empirically measure the behaviour of the system under increased load were conducted. This revealed that when under simulated expected heavy loads, the application instances bore the brunt of the extra work, thereby demonstrating that the system would be able to handle projected future workloads as anticipated, and that extra application instances could be provisioned to increase capacity.

Additionally, due to the use of caches to retain the structure of the curriculum, and the architecture of the application more generally, the workload when the curriculum visible to students is not being changed is remarkably light, even when many students are navigating the site. This is because the browser loads the curriculum and associated data as the student logs in, then makes no further requests while the student is navigating the site.

While these tests indicated that the system would behave well under projected workloads, these were simulated based on expected load. The monitoring described above will provide insight into the achieved performance of the system as additional institutions are added. With a view even further ahead, monitoring will be used to inform decisions regarding any future architectural alterations that may be required.

3.9.8 Maintenance and upgrades

3.9.8.1 Maintenance windows for AWS hosted services

The hosted AWS data storage servers have been configured to allow a weekly maintenance between 2-3 am UTC every Tuesday. During this window, AWS will apply any required updates to the underlying operating system and the service running.

Further, for services configured to have a primary with a replica, the update is first applied to the replica, which is then promoted to become the new primary. The old primary is then updated and becomes the new replica.

It is not anticipated that these will impact the delivery of the system.

3.9.8.2 Software and infrastructure upgrades

Agreement for any changes that may impact the use of the system are agreed before being enacted. Further, all software updates to the production environment are currently communicated and agreed prior to deployment.

4 Security

4.1 Design features

4.1.1 Partitioning of environments

Environments are separated by the use of Kubernetes namespaces, and by use of separate AWS IAM roles used between the environments to ensure no privilege escalation between environments or accidental promotion of software to the wrong environment.

4.1.2 Partitioning of components

Each component is partitioned as much as is possible using AWS features. Access to each component can then be individually controlled.

4.1.3 Use of encryption

TLS 1.2 is used for connections from browsers to CloudFront, and any attempts to connect using plain HTTP are redirected.

No encryption is used over the network between the internal components of the system. AWS describes the VPC as being “logically isolated”, and it is treated as a private network.

4.1.4 Logging

All system logs from EC2 machines are sent to CloudWatch Logs, where they are accessible to privileged accounts only. This helps isolate systems logs in the case of suspicious activity.

Operational protocols for the identification of suspicious activity have yet to be defined.

4.1.5 User accounts

In normal operation no access to any EC2 nodes is required for any maintenance or operational purpose. Any access to a user account is therefore a potential compromise and can be alerted appropriately.

4.2 Penetration testing

Penetration tests will be conducted following every major release, and results of the latest test will be available on request.

These tests will be run on the stage environment, running the same code and with the same architecture as the production environment, but will not impact the service for production users, or risk altering production data.

4.3 Confidential data

The system uses Amazon RDS as a primary data store, in which user data is stored at rest. This data includes the first name, last name, email address and hashed password of any user who does not authenticate using SSO.

The “intake” spreadsheet uploads used to create and update student user records and curriculum steps are retained in the site-specific S3 bucket for audit purposes, but these files are only available to operations staff, not through the application.

4.4 Relevant use cases

Use Case	**Description**
Student data upload	Gatekeepers access the site over TLS encrypted connections only. They log in using SSO. Spreadsheet uploads are processed and stored securely.
Student browse and explore	Students access the site over TLS encrypted connections only. They log in using SSO. They browse the curriculums they have access to.
Student calendar	Students see calendar data they have access to, and may create personal events only visible to themselves.
Student notes	Students make notes and upload files which are stored securely and only visible to themselves.
Student revision	Students may mark learning outcomes as achieved, only visible to themselves.
Student leaves feedback	Students submit feedback for later review by editors.
Student feedback download	Editors view and download feedback including data on the student who submitted the feedback.
Curriculum maintenance	Editors make changes to the curriculum and upload/replace/delete resources. A revision history is retained of curriculum changes.
Curriculum update and rollover	Gatekeepers initiate the ‘rollover’ process, cloning a curriculum and its resources and making the previous “upcoming” curriculum available to students.

Curriculum import/export	Import and export functionality is available to operators with SSH access to the control server or in the front end to users with specific permission.

4.5 System threat model

4.5.1 Entry points

4.5.1.1 AWS Control Panel

Using the AWS control panel every aspect of the system can be amended or controlled. In particular a database snapshot can be taken and downloaded. Access to the Control Panel is secured using strong passwords and two-factor authentication.

4.5.1.2 AWS API

The API has the same features as the control panel, although different authentication keys are used.

4.5.1.3 EKS IAM Roles for Service Accounts

EKS injects configuration into containers running within the cluster to grant access to AWS resources. The permissions granted include access to some privileged information, particularly the RDS password. If this service were breached, this information could be obtained.

4.5.1.4 Elastic Load Balancer

The Elastic Load Balancer provides the only Internet-accessible part of the service in production, on port 80. The ELB has network access to the application servers over port 80. Security groups are used to restrict access, limited to the IPs used by CloudFront.

4.5.1.5 CloudFront

CloudFront hosts TLS certificates for the public domains, and holds cached public assets (CSS, JavaScript, icons, fonts, logos).

4.5.1.6 The web application

The web application and its API could include any of a number of vulnerabilities – mitigations for these are discussed later.4.5.1.8 AWS Infrastructure

AWS itself provides infrastructure such as network

connectivity and hypervisors. A breach of the AWS hosting environment would provide access to any or perhaps all of the application.

4.5.2 Assets

4.5.2.1 Scope

The scope of confidential information here is limited to personally identifiable information such as:

- Name
- Email Address
- Password (unless user only authenticates with SSO)

Further, the contents of the curriculum should be considered confidential, as well as student feedback, notes and file attachments.

4.5.2.2 System logs

System logs may contain confidential information, depending on the nature of errors, backtraces etc.

4.5.2.3 Backups

Database backups are stored by the RDS service. These will contain confidential personal information.

4.5.2.4 Relational database service (RDS)

RDS hosts all confidential information.

4.5.2.5 ElastiCache

The cache stores curriculum data, but not user data.

The cache will contain confidential information in RAM only. If the RAM of the cache system were accessed then this could be breached.

4.5.3 Trust levels

4.5.3.1 AWS staff

Amazon staff are not specifically granted any access as part of this system. Nevertheless they have potential access to any part of the system.

4.5.3.2 Operations

Operations staff with AWS access have complete control over the system and its functions.

4.5.3.3 Developers

Developers are able to commit code. The lead developer is able to merge code, build and deploy AMIs.

4.5.3.4 Gatekeepers

These users have access to system tools – specifically the list of users – and therefore access to personal data.

4.5.3.4 Editors

Editors have some privileged access, in particular can download feedback which includes personal data.

4.5.3.5 Students

Students have access only to their curriculums, and their own notes and personal files they have uploaded.

4.5.4 Risks

The list of risks below is taken from the OWASP 2021 Top 10 most critical web application security risks report. This has been augmented with any other risks identified as being relevant to the architecture or software.

Risk	Description	Assessment	Rank
OWASP 1: Broken access control	Applications need to perform access checks on the server side as well as client side.	A common problem with web applications. Mitigated with the use of permissions and validation support within Django.	MEDIUM
OWASP 2: Cryptographic failures	Sensitive data should be protected with encryption, and special precautions taken when interacting with the browser.	Personal data is transferred between several components.	MEDIUM
OWASP 3: Injection	Injection flaws such as SQL, XSS, OS and LDAP injection.	All RDBMS access is mediated by an ORM. No OS level commands are executed. Web content is entered by administrative users or personal to the contributor. Content is filtered to specific tags. Script origin is limited by Content Security Policy.	LOW
OWASP 4: Insecure design	Software and infrastructure design should work to prevent common vulnerabilities	Design follows a common reference design pattern. Main risk is around multi-tenancy being enforced in code.	LOW
OWASP 5: Security misconfiguration	Misconfigurations can expose confidential data, or allow access it should not.	Security configuration for this system is reasonably complex.	MEDIUM

OWASP 6: Vulnerable and Outdated Components	Components should be monitored for known vulnerabilities and upgraded or patched.	Operating system components maintained by AWS. Python and JavaScript components may have vulnerabilities.	MEDIUM
OWASP 7: Identification and Authentication Failures	Implementation errors in authentication or session management allow an attack to assume another user's identity.	Authentication and session handling is provided by Django, and is well proven.	LOW
OWASP 8: Software and Data Integrity Failures	Vulnerable deserializers could be exploited if an attacker sends malicious data.	Data submitted by authenticated users is deserialized on the server. Particular risk for external XML entities via SSO.	HIGH
OWASP 9: Security Logging and Monitoring Failures	Should any kind of incident occur, failure to detect and respond could allow the result to be far more severe.	Reasonably diverse system architecture to monitor, and a few different sources of logs.	MEDIUM
OWASP 10: Server Side Request Forgery	Server must not fetch a remote resource from unvalidated user input.	Code review to ensure appropriate handling of inputs prior to the server making requests.	LOW
Cross Site Request Forgery	A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application.	Anti-CSRF measures are provided by Django, and are well proven.	LOW
Unauthorised access to AWS API or Control Panel	Anyone with privileged access to these services can do anything.	Significant threat. Must be mitigated with good practice.	HIGH
AWS virtualization breach	In theory a breach of the virtualization environment provides access to the RAM of worker and application servers.	Existential threat to AWS business as a whole.	LOW
AWS insider	An AWS employee with access to infrastructure could do anything.	Existential threat to AWS business as a whole. Low likelihood.	LOW
Unauthorised access to system tools	Access with correct privilege levels would allow access to user data.	Significant threat. Should be closely controlled and monitored.	HIGH
Unauthorised SSH access to servers	With correct credentials, user could access RAM of all worker and application servers	Significant threat. Can be mitigated by removing all SSH access.	HIGH

Tenancy leakage	Misconfiguration or error could allow data belonging to one tenant to be inappropriately mixed with data from another and displayed.	High impact fault.	HIGH

4.5.5 Mitigation

4.5.5.1 Standard quality procedures

Many of the above risks are mitigated by the suite of standard quality techniques implemented by the development team, including:

- Appropriate staff training
- Code review
- Automated unit testing
- Automated integration testing
- Automated system testing
- Separate QA function, qualified to industry standards
- Manual testing of high risk areas
- Use of analysis and design approaches
- Regular use of AWS's "Trusted Advisor" tool to identify any issues

4.5.5.2 Specific mitigations for High and Medium risks

Risk	**Mitigation approach**
OWASP 1: Broken access control	1. Code review 2. Vulnerability scanning
OWASP 2: Cryptographic failures	1. Use of encryption over public network, and data at rest
OWASP 5: Security misconfiguration	1. Limiting manual changes to security as part of normal course of business 2. Change control of all security configuration 3. Automated ongoing test pack monitoring security quality attributes
OWASP 6: Vulnerable and Outdated Components	1. Automated patching process 2. Vulnerability monitoring and review

OWASP 8: Software and Data Integrity Failures	1. Review of all parts of the system that deserialize data from users, particularly XML from SSO 2. Vulnerability monitoring and review 3. Code review
OWASP 9: Security Logging and Monitoring Failures	1. Use of anomaly detection on relevant metrics within monitoring 2. Aggregated logging
Unauthorised access to AWS API or Control Panel	1. Limited number of users with access 2. Control of credentials 3. Use of two-factor authentication
Unauthorised access to system tools	1. Control of credentials
Unauthorised SSH access to control server	1. Limited number of users with access 2. Limited IP addresses with access 3. Control of credentials
Tenancy leakage	1. Constraints in data access layer 2. Automated testing