

Module Interface Specification for Software Engineering

Team 21, Visionaries

Angela Zeng

Ann Shi

Ibrahim Sahi

Manan Sharma

Stanley Chen

January 22, 2026

1 Revision History

Date	Version	Notes
Nov 13	1.0	Version 1 completed

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at <https://github.com/mansha71/CapstoneProject/tree/main/docs/SRS>

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	Introduction	1
4	Notation	1
4.1	Derived Types	1
5	Module Decomposition	3
6	MIS of Hardware-Hiding Module	4
6.1	Module	4
6.2	Uses	4
6.3	Syntax	4
6.3.1	Exported Constants	4
6.3.2	Exported Access Programs	4
6.4	Semantics	4
6.4.1	State Variables	4
6.4.2	Environment Variables	5
6.4.3	Assumptions	5
6.4.4	Access Routine Semantics	5
6.4.5	Local Functions	6
7	MIS of Data Ingestion Module	6
7.1	Module	6
7.2	Uses	6
7.3	Syntax	6
7.3.1	Exported Constants	6
7.3.2	Exported Access Programs	6
7.4	Semantics	6
7.4.1	State Variables	6
7.4.2	Environment Variables	6
7.4.3	Assumptions	7
7.4.4	Access Routine Semantics	7
7.4.5	Local Functions	8
8	MIS of Real-Time Streaming Module	9
8.1	Module	9
8.2	Uses	9
8.3	Syntax	9
8.3.1	Exported Constants	9

8.3.2	Exported Access Programs	9
8.4	Semantics	9
8.4.1	State Variables	9
8.4.2	Environment Variables	9
8.4.3	Assumptions	10
8.4.4	Access Routine Semantics	10
8.4.5	Local Functions	10
9	MIS of Data Preprocessing Module	11
9.1	Module	11
9.2	Uses	11
9.3	Syntax	11
9.3.1	Exported Constants	11
9.3.2	Exported Access Programs	11
9.4	Semantics	12
9.4.1	State Variables	12
9.4.2	Environment Variables	12
9.4.3	Assumptions	12
9.4.4	Access Routine Semantics	12
9.4.5	Local Functions	13
10	MIS of Privacy Filtering Module	14
10.1	Module	14
10.2	Uses	14
10.3	Syntax	14
10.3.1	Exported Constants	14
10.3.2	Exported Access Programs	14
10.4	Semantics	14
10.4.1	State Variables	14
10.4.2	Environment Variables	15
10.4.3	Assumptions	15
10.4.4	Access Routine Semantics	15
10.4.5	Local Functions	15
11	MIS of Access Control Module	16
11.1	Module	16
11.2	Uses	16
11.3	Syntax	16
11.3.1	Exported Constants	16
11.3.2	Exported Access Programs	16
11.4	Semantics	16
11.4.1	State Variables	16
11.4.2	Environment Variables	16

11.4.3	Assumptions	17
11.4.4	Access Routine Semantics	17
11.4.5	Local Functions	17
12	MIS of Secure Storage & Retention Module	18
12.1	Module	18
12.2	Uses	18
12.3	Syntax	18
12.3.1	Exported Constants	18
12.3.2	Exported Access Programs	18
12.4	Semantics	18
12.4.1	State Variables	18
12.4.2	Environment Variables	18
12.4.3	Assumptions	19
12.4.4	Access Routine Semantics	19
12.4.5	Local Functions	20
13	MIS of Observability Module	21
13.1	Module	21
13.2	Uses	21
13.3	Syntax	21
13.3.1	Exported Constants	21
13.3.2	Exported Access Programs	21
13.4	Semantics	21
13.4.1	State Variables	21
13.4.2	Environment Variables	21
13.4.3	Assumptions	21
13.4.4	Access Routine Semantics	22
13.4.5	Local Functions	22
14	MIS of Engagement Analytics Module	23
14.1	Module	23
14.2	Uses	23
14.3	Syntax	23
14.3.1	Exported Constants	23
14.3.2	Exported Access Programs	23
14.4	Semantics	23
14.4.1	State Variables	23
14.4.2	Environment Variables	23
14.4.3	Assumptions	24
14.4.4	Access Routine Semantics	24
14.4.5	Local Functions	24

15 MIS of Correlation and Visual Analysis Module	25
15.1 Module	25
15.2 Uses	25
15.3 Syntax	25
15.3.1 Exported Constants	25
15.3.2 Exported Access Programs	25
15.4 Semantics	25
15.4.1 State Variables	25
15.4.2 Environment Variables	25
15.4.3 Assumptions	26
15.4.4 Access Routine Semantics	26
15.4.5 Local Functions	26
16 MIS of Dashboard Visualization Module	27
16.1 Module	27
16.2 Uses	27
16.3 Syntax	27
16.3.1 Exported Constants	27
16.3.2 Exported Access Programs	27
16.4 Semantics	27
16.4.1 State Variables	27
16.4.2 Environment Variables	27
16.4.3 Assumptions	28
16.4.4 Access Routine Semantics	28
16.4.5 Local Functions	28
17 MIS of Reporting Module	29
17.1 Module	29
17.2 Uses	29
17.3 Syntax	29
17.3.1 Exported Constants	29
17.3.2 Exported Access Programs	29
17.4 Semantics	29
17.4.1 State Variables	29
17.4.2 Environment Variables	29
17.4.3 Assumptions	30
17.4.4 Access Routine Semantics	30
17.4.5 Local Functions	30

3 Introduction

The following document details the Module Interface Specifications for Visionaries, a system designed to analyze and visualize student engagement during lectures using eye-tracking technology. Each module is described in terms of its purpose, syntax, semantics, and interactions with other modules.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at <https://github.com/mansha71/CapstoneProject>.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from ?. The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol $:=$ is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Software Engineering.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	\mathbb{Z}	a number without a fractional component in $(-\infty, \infty)$
natural number	\mathbb{N}	a number without a fractional component in $[1, \infty)$
real	\mathbb{R}	any number in $(-\infty, \infty)$

The specification of Software Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Software Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

4.1 Derived Types

The following derived types are used throughout the module specifications:

Type Name	Definition	Description
DeviceID	String	Unique identifier for tracking device
SessionID	String	Unique identifier for session
PseudonymID	String	Pseudonymous DeviceID
DataBlob	ByteArray	Raw binary data
StreamConfig	Tuple(host: String, port: \mathbb{N} , bufferSize: \mathbb{N})	Configuration for stream
StreamSession	Tuple(sessionID: SessionID, lastFrame: VideoFrame, timestamp: \mathbb{R})	Active session state
VideoFrame	ByteArray	Single frame of video
SystemMetrics	Tuple(cpuUsage: \mathbb{R} , memoryUsage: \mathbb{R} , activeStreams: \mathbb{N})	Runtime performance metrics
SystemEvent	Tuple(eventType: String, timestamp: \mathbb{R} , details: String)	Logged system event
RetentionPolicy	Tuple(maxAgeDays: \mathbb{N} , maxSessions: \mathbb{N})	Rules for session retention
StorageRef	String	File path reference to data
StreamData	Sequence(VideoFrame)	Sequence of video frames
Role	String	Role label used for instructor, researcher, etc.
ResourceID	String	Identifier for a resource, session, report, experiment, etc.
Status	String	Return status like opened, closed, etc.
PrivacyEvent	Tuple(eventType: String, deviceID: DeviceID, timestamp: \mathbb{R})	Privacy-related event
AccessEvent	Tuple(role: Role, resource: ResourceID, timestamp: \mathbb{R})	Authorization/access event
SystemReport	Tuple(health: String, summary: String, timestamp: \mathbb{R})	Summary view of system state

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	Level 2
Hardware-Hiding Module	Hardware-Hiding Module
Behaviour-Hiding Modules	Data Ingestion Module Real-Time Streaming Module Dashboard Visualization Module Reporting Module
Software Decision Modules	Data Preprocessing Module Privacy Filtering Module Access Control Module Secure Storage & Retention Module Observability Module Engagement Analytics Module Correlation & Visual Analysis Module

Table 1: Module Hierarchy for Visionaries System

6 MIS of Hardware-Hiding Module

6.1 Module

This module abstracts the physical sensing and I/O devices (eye-tracking glasses, cameras, microphones, storage interfaces, and network interfaces) behind a uniform virtual device interface. Downstream modules interact only with this interface and do not depend on vendor SDK calls, device driver details, or OS-specific mechanisms.

6.2 Uses

None.

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	Input	Output	Exceptions
openDevice	deviceId: DeviceID	status: Status	ConnectionError
closeDevice	deviceId: DeviceID	status: Status	None
readGazeFrame	deviceId: DeviceID	rawGazeData: DataBlob	ConnectionError
readWorldFrame	deviceId: DeviceID	rawVideoData: VideoFrame	ConnectionError
readAudioChunk	deviceId: DeviceID	rawAudioData: DataBlob	ConnectionError
getDeviceMetadata	deviceId: DeviceID	metadata: DataBlob	None

Table 2: Exported Access Programs for Hardware-Hiding Module

6.4 Semantics

6.4.1 State Variables

openDevices: Set(DeviceID)

Tracks currently opened/available devices.

6.4.2 Environment Variables

Vendor SDKs, OS device drivers, and hardware interfaces.

6.4.3 Assumptions

Devices are discoverable by the underlying OS/SDK and can be opened/closed by the process.

6.4.4 Access Routine Semantics

openDevice(deviceID: DeviceID):

- transition: $\text{openDevices} := \text{openDevices} \cup \{\text{deviceID}\}$
- output: $\text{status} := \text{opened}$
- exception: Raises ConnectionError if the device cannot be opened.

closeDevice(deviceID: DeviceID):

- transition: $\text{openDevices} := \text{openDevices} \setminus \{\text{deviceID}\}$
- output: $\text{status} := \text{closed}$
- exception: None

readGazeFrame(deviceID: DeviceID):

- transition: None
- output: rawGazeData
- exception: Raises ConnectionError if the device is unavailable.

readWorldFrame(deviceID: DeviceID):

- transition: None
- output: rawVideoData
- exception: Raises ConnectionError if the device is unavailable.

readAudioChunk(deviceID: DeviceID):

- transition: None
- output: rawAudioData
- exception: Raises ConnectionError if the device is unavailable.

getDeviceMetadata(deviceID: DeviceID):

- transition: None
- output: metadata
- exception: None

6.4.5 Local Functions

sdkRead: DeviceID \rightarrow DataBlob

Semantics: Reads a device-specific payload via vendor SDK/driver calls.

7 MIS of Data Ingestion Module

7.1 Module

This module entails collecting eye-tracking and video data from multiple eye-tracking devices, video data from the central camera and screen recording, as well as instructor audio recordings and pre-/post-lecture questionnaire responses. It is responsible solely for the reliable capture and initial storage of all incoming raw data streams, without performing any preprocessing or filtering operations.

7.2 Uses

Hardware-Hiding Module

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Access Programs

7.4 Semantics

7.4.1 State Variables

storedData: list of captured raw data streams currently saved in local or cloud storage.

7.4.2 Environment Variables

Connections to Pupil Labs Neon devices, central camera, screen-recording source, audio input, and questionnaire input sources.

Name	Input	Output	Exceptions
collectGazeData	deviceId, streamConfig	rawGazeData	ConnectionError
collectVideoData	sourceID, streamConfig	rawVideoData	ConnectionError
collectAudioData	deviceId	rawAudioData	ConnectionError
collectQuestionnaire	formInput	questionnaireData	FormatError
storeRawData	dataBundle	status	IOError

Table 3: Exported Access Programs for Data Ingestion Module

7.4.3 Assumptions

All recording devices are properly connected, network latency is within acceptable limits for real-time transfer, and the file system or storage backend is accessible.

7.4.4 Access Routine Semantics

collectGazeData():

- transition: Initiates collection of gaze and world-view video data from connected eye-tracking devices.
- output: Returns raw gaze and eye-camera data.
- exception: Raises ConnectionError if device is unavailable.

collectVideoData():

- transition: Records video data from the central camera and screen-capture feed.
- output: Returns raw video frames.
- exception: Raises ConnectionError if any video source is unavailable.

collectAudioData():

- transition: Captures instructor audio from connected microphone devices.
- output: Returns raw audio stream data.
- exception: Raises ConnectionError if the audio device is unavailable.

collectQuestionnaire():

- transition: Collects pre- or post-lecture questionnaire submissions.
- output: Returns questionnaire data.

- exception: Raises `FormatError` for incomplete or malformed responses.

`storeRawData()`:

- transition: Saves all raw data streams to secure storage with metadata describing device, timestamp, and session.
- output: Returns success status.
- exception: Raises `IOError` if data cannot be written.

7.4.5 Local Functions

`validateConnection`: `DeviceID` \rightarrow `Boolean`

`verifySourceAvailability`: `String` \rightarrow `Boolean`

8 MIS of Real-Time Streaming Module

8.1 Module

This module manages real-time transmission of gaze and video frames from multiple capture devices to a single dashboard client. It supports concurrent device streams and ensures low-latency delivery for live visualization.

8.2 Uses

Data Ingestion Module, Data Preprocessing Module, Privacy Filtering Module, Access Control Module, Secure Storage & Retention Module, Observability Module

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	Input	Output	Exceptions
initializeStream	config: StreamConfig	sessionID	ConnectionError
transmitData	deviceID: DeviceID, frame: VideoFrame	status	StreamError
terminateStream	sessionID: SessionID	status	None

Table 4: Exported Access Programs for Real-Time Streaming Module

8.4 Semantics

8.4.1 State Variables

activeSessions: Map(DeviceID \rightarrow StreamSession)

Stores the active streaming session information for each connected device.

8.4.2 Environment Variables

Network socket interface for dashboard communication.

8.4.3 Assumptions

A stable network connection exists and the dashboard client is available to receive data. Only live streaming is supported; no persistent buffering, recording, or replay functionality is implemented by this module.

8.4.4 Access Routine Semantics

`initializeStream(config: StreamConfig):`

- transition: Initializes an empty session map and prepares network communication resources.
- output: `sessionID`
- exception: `ConnectionError`

`transmitData(deviceID: DeviceID, frame: VideoFrame):`

- transition: `activeSessions[deviceID].lastFrame := frame`
- output: `status := success`
- exception: `StreamError`

`terminateStream(sessionID: SessionID):`

- transition: `activeSessions := \emptyset`
- output: `status := terminated`
- exception: `None`

8.4.5 Local Functions

`validateFrame: VideoFrame \rightarrow Boolean`

`openSocket: StreamConfig \rightarrow NetworkHandle`

9 MIS of Data Preprocessing Module

9.1 Module

This module prepares the captured data for analysis by filtering noise, normalizing coordinates, and structuring the gaze and video data into usable formats. The recording inputs from the eye-tracking devices, central camera, and screen-recording coordinates are synchronized for homography, and instructor coordinates are generated throughout the recording. The screen-recording input is categorized based on visual item classes (e.g., text, diagram, image regions), and the audio stream is denoised, segmented, and temporally aligned with the synchronized video data. Questionnaire responses are cleaned by removing empty, duplicate, or invalid entries. This ensures all data sources are aligned, validated, and ready for analysis in downstream modules.

9.2 Uses

Data Ingestion Module

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	Input	Output	Exceptions
filterNoise	rawData	cleanData	DataError
synchronizeStreams	cleanDataBundle	syncedData	SyncError
computeHomography	syncedVideoData	alignedData	MathError
categorizeVisualItems	screenRecording	categorizedSegments	FormatError
cleanAudioStream	rawAudioData	processedAudio	AudioError
cleanQuestionnaireResponses	rawQuestionnaireData	validatedQuestionnaire	DataError
formatForAnalysis	alignedData	structuredData	FormatError

Table 5: Exported Access Programs for Data Preprocessing Module

9.4 Semantics

9.4.1 State Variables

None. Operates statelessly on provided input data.

9.4.2 Environment Variables

None.

9.4.3 Assumptions

Input data follows the expected format, timestamps are available for all streams, and synchronization metadata is provided from the ingestion stage.

9.4.4 Access Routine Semantics

`filterNoise(rawData):`

- transition: Applies filters (e.g., smoothing, low-pass, temporal averaging) to remove noise from gaze, video, and audio streams.
- output: `cleanData`
- exception: Raises `DataError` for corrupted or incomplete data.

`synchronizeStreams(cleanDataBundle):`

- transition: Aligns gaze, video, screen recording, and audio streams using timestamps or synchronization cues.
- output: `syncedData`
- exception: Raises `SyncError` if synchronization fails.

`computeHomography(syncedVideoData):`

- transition: Computes spatial mappings between gaze coordinates and screen coordinates.
- output: `alignedData`
- exception: Raises `MathError` if homography computation fails.

`categorizeVisualItems(screenRecording):`

- transition: Segments screen recordings into visual item classes such as text, diagrams, and images.

- output: categorizedSegments
- exception: Raises FormatError if segmentation fails.

cleanAudioStream(rawAudioData):

- transition: Removes background noise and aligns audio with synchronized video streams.
- output: processedAudio
- exception: Raises AudioError if audio processing fails.

cleanQuestionnaireResponses(rawQuestionnaireData):

- transition: Removes empty, duplicate, or malformed questionnaire responses.
- output: validatedQuestionnaire
- exception: Raises DataError if validation fails.

formatForAnalysis(alignedData):

- transition: Structures synchronized and aligned data into a standardized schema.
- output: structuredData
- exception: Raises FormatError for invalid schema generation.

9.4.5 Local Functions

interpolateMissingFrames: StreamData \rightarrow StreamData

alignTimestamps: Sequence(DataBlob) \rightarrow Sequence(DataBlob)

extractInstructorCoordinates: VideoFrame \rightarrow CoordinateStream

computeVisualClasses: VideoFrame \rightarrow VisualLabel

10 MIS of Privacy Filtering Module

10.1 Module

This module applies privacy filtering to sensitive identifiers before data is stored, streamed, or exported. The implemented anonymization is pseudonymous: device identifiers are replaced using a deterministic pseudonymization rule. Consent collection is handled externally; this module may verify consent status if provided by the calling context.

10.2 Uses

Data Ingestion Module, Data Preprocessing Module

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

Name	Input	Output	Exceptions
generatePseudonym	deviceId: DeviceID	pseudonymID: PseudonymID	PrivacyError
anonymizeParticipant	deviceId: DeviceID	anonymizedID: PseudonymID	PrivacyError
verifyConsent	deviceId: DeviceID	isConsented: Boolean	None

Table 6: Exported Access Programs for Privacy Filtering Module

10.4 Semantics

10.4.1 State Variables

pseudonymMap: $\text{Map}(\text{DeviceID} \rightarrow \text{PseudonymID})$

Stores the pseudonymous identifier associated with each device identifier for the current execution context.

consentMap: $\text{Map}(\text{DeviceID} \rightarrow \text{Boolean})$

Tracks whether a participant (identified by deviceId) has provided consent.

10.4.2 Environment Variables

None.

10.4.3 Assumptions

Participant consent is handled externally, but consent status may be provided and tracked within this module for runtime verification.

Pseudonym mappings are accessible only within controlled internal contexts and are not exposed to external systems or users.

This module does not implement irreversible anonymization beyond pseudonymization.

10.4.4 Access Routine Semantics

generatePseudonym(deviceID: DeviceID):

- transition:
 - If $\text{deviceID} \notin \text{dom}(\text{pseudonymMap})$, then $\text{pseudonymMap}[\text{deviceID}] := \text{newPseudonym}$.
- output: $\text{pseudonymID} := \text{pseudonymMap}[\text{deviceID}]$
- exception: Raises `PrivacyError` if a pseudonym cannot be generated for the provided deviceID.

anonymizeParticipant(deviceID: DeviceID):

- transition:
 - Ensures a pseudonym exists for deviceID by invoking `generatePseudonym(deviceID)` when needed.
- output: $\text{anonymizedID} := \text{pseudonymMap}[\text{deviceID}]$
- exception: Raises `PrivacyError` if the deviceID cannot be mapped to a pseudonym.

verifyConsent(deviceID: DeviceID):

- transition: None
- output: $\text{isConsented} := \text{consentMap}[\text{deviceID}]$ (defaults to false if not present)
- exception: None

10.4.5 Local Functions

generateSecureID: DeviceID \rightarrow PseudonymID

Semantics: Produces a pseudonymous identifier based on the provided DeviceID using a deterministic rule.

11 MIS of Access Control Module

11.1 Module

This module enforces authorization decisions for system actions such as viewing dashboards, subscribing to streams, accessing stored sessions, and exporting reports. It encapsulates role-based access control (RBAC) policy and exposes authorization checks to other modules. This module does not implement privacy anonymization or consent collection. Security-relevant authorization events are emitted to the Observability Module, which owns audit logging policy and storage.

11.2 Uses

Observability Module

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	Input	Output	Exceptions
validateAccess	userRole: Role, resource: ResourceID	permissionStatus: AuthError Boolean	
logAccess	event: AccessEvent	status: Status	None

Table 7: Exported Access Programs for Access Control Module

11.4 Semantics

11.4.1 State Variables

None. Authorization decisions are computed from provided role and resource inputs. Audit records are emitted to the Observability Module.

11.4.2 Environment Variables

None.

11.4.3 Assumptions

Roles are provided by the calling context and are already authenticated externally. Authorization policy is defined as role-to-resource permissions and may evolve independently of other system modules.

11.4.4 Access Routine Semantics

validateAccess(userRole: Role, resource: ResourceID):

- transition:
 - permissionStatus := (userRole is permitted to access resource)
 - Emit a SystemEvent describing the authorization decision to the Observability Module
- output: permissionStatus
- exception: Raises AuthError if the userRole or resource identifier is invalid.

logAccess(event: AccessEvent):

- transition: Convert the AccessEvent to a SystemEvent and emit it to the Observability Module
- output: status := logged
- exception: None

11.4.5 Local Functions

createAuthEvent: (Role, ResourceID, Boolean) → SystemEvent

Semantics: Creates a structured authorization audit event describing an allow or deny decision.

convertAccessEvent: AccessEvent → SystemEvent

Semantics: Transforms an AccessEvent into a SystemEvent suitable for logging by Observability.

12 MIS of Secure Storage & Retention Module

12.1 Module

This module is responsible for storing and retrieving session artifacts and enforcing retention and deletion policies. Data is stored locally as files and may be accessed later. When `retain=true`, session data is persisted to disk; when `retain=false`, data is used for live processing only and no disk persistence occurs.

12.2 Uses

Data Ingestion Module, Data Preprocessing Module, Privacy Filtering Module

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	Input	Output	Exceptions
<code>storeSessionData</code>	<code>sessionID: SessionID</code> , <code>data: DataBlob</code> , <code>retain: Boolean</code>	<code>status: Status</code>	<code>IOException</code>
<code>retrieveSessionData</code>	<code>sessionID: SessionID</code>	<code>data: DataBlob</code>	<code>IOException</code>
<code>deleteSessionData</code>	<code>sessionID: SessionID</code>	<code>status: Status</code>	<code>IOException</code>
<code>enforceRetentionPolicy</code>	<code>policy: RetentionPolicy</code>	<code>status: Status</code>	None

Table 8: Exported Access Programs for Secure Storage & Retention Module

12.4 Semantics

12.4.1 State Variables

`storedSessions: Map(SessionID → StorageRef)`

Maps each stored session to a storage reference (for example, a file path).

12.4.2 Environment Variables

Local file system access for reading and writing session artifacts.

12.4.3 Assumptions

Data retention is controlled by a boolean retain flag supplied by the caller.

Storage is local-only (no cloud backend assumed).

If retain = false, data may be used live but is not persisted by this module.

12.4.4 Access Routine Semantics

storeSessionData(sessionID: SessionID, data: DataBlob, retain: Boolean):

- transition:
 - If retain = true, write data to local storage and set storedSessions[sessionID] := storageRef.
 - If retain = false, data is used for live processing only; no disk persistence occurs and storedSessions is unchanged.
- output: status := success if retain = false or if the write succeeds; otherwise raises an exception
- exception: Raises IOError if retain = true and data cannot be written.

retrieveSessionData(sessionID: SessionID):

- transition: None
- output: data := read data from storedSessions[sessionID]
- exception: Raises IOError if sessionID is not found in storedSessions or the storage reference cannot be read.

deleteSessionData(sessionID: SessionID):

- transition:
 - Remove persisted artifacts for sessionID from storage (if present).
 - Remove sessionID from storedSessions if it exists.
- output: status := deleted
- exception: Raises IOError if deletion fails due to file system errors.

enforceRetentionPolicy(policy: RetentionPolicy):

- transition: Deletes stored sessions that violate the provided retention policy.
- output: status := applied
- exception: None

12.4.5 Local Functions

resolveStorageRef: SessionID \rightarrow StorageRef

Semantics: Determines the storage reference used for a given session.

13 MIS of Observability Module

13.1 Module

This module handles system metrics collection, stream health monitoring, and error/event logging for runtime observability. It is responsible for reporting basic health information about the system execution and recording significant runtime events.

13.2 Uses

None.

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	Input	Output	Exceptions
monitorInfrastructure	metrics: SystemMetrics	systemReport: SystemReport	None
logSystemEvent	event: SystemEvent	status: Status	None

Table 9: Exported Access Programs for Observability Module

13.4 Semantics

13.4.1 State Variables

systemEventLog: Sequence(SystemEvent)
systemMetricsLog: Sequence(SystemMetrics)

13.4.2 Environment Variables

System monitoring interfaces available to the runtime environment.

13.4.3 Assumptions

Metrics are best-effort: missing metrics do not crash the system.
This module does not enforce alerting or external monitoring integrations unless added later.

13.4.4 Access Routine Semantics

monitorInfrastructure(metrics: SystemMetrics):

- transition: systemMetricsLog := systemMetricsLog \cup {metrics}
- output: systemReport := summary of current metrics (for example, recent values and basic health status)
- exception: None

logSystemEvent(event: SystemEvent):

- transition: systemEventLog := systemEventLog \cup {event}
- output: status := logged
- exception: None

13.4.5 Local Functions

summarizeMetrics: Sequence(SystemMetrics) \rightarrow SystemReport

Semantics: Produces a SystemReport summarizing recent system metrics.

14 MIS of Engagement Analytics Module

14.1 Module

This module analyzes student engagement by combining attention metrics with pre- and post-lecture questionnaire results. Questionnaire data is collected through the Data Ingestion Module, anonymized by the Privacy Filtering Module and preprocessed before reaching this module. Each question is associated with the content of a specific slide, helping compute slide-level learning scores. With pre- and post-lecture questionnaires, changes in student understanding can be identified and correlated with gaze-data. These analytics quantify the effectiveness of each slide, object type, instructor-directed attention, and instructor audio trends.

14.2 Uses

Data Preprocessing Module, Privacy Filtering Module

14.3 Syntax

14.3.1 Exported Constants

None.

14.3.2 Exported Access Programs

Name	Input	Output	Exceptions
computeLearningScores	anonymizedQuestionnaireData, slideMap	LearningScores	DataError

Table 10: Exported Access Programs for Engagement Analytics Module

14.4 Semantics

14.4.1 State Variables

None.

14.4.2 Environment Variables

None.

14.4.3 Assumptions

Questionnaire data has already been collected, anonymized, and validated. Each questionnaire item includes a slide reference.

14.4.4 Access Routine Semantics

`computeLearningScores(anonymizedQuestionnaireData, slideMap):`

- **transition:** Computes pre/post learning gains per slide using questionnaire data linked to slide identifiers. Aggregates multiple questions associated with the same slide.
- **output:** `learningScores`
- **exception:** Raises `DataError` for missing, malformed, or improperly mapped questionnaire items.

14.4.5 Local Functions

`computeLearningDelta()`, `aggregateSlideScores()`

15 MIS of Correlation and Visual Analysis Module

15.1 Module

This module maps gaze data into pixel-based scene coordinates, detects slide transitions from live video frames, classifies gaze targets, and aligns learning analytics with visual content.

15.2 Uses

Data Preprocessing Module, Engagement Analytics Module

15.3 Syntax

15.3.1 Exported Constants

None.

15.3.2 Exported Access Programs

Name	Input	Output	Exceptions
mapCoordinatesToScene	worldViewData, sceneModel	pixelMappedData	MappingError
detectSlideTransitions	liveVideoFrames	slideTimeline	VideoError
classifyGazeTarget	pixelMappedData, slideOb- jects, instructorRegions	labeledGazeData	ClassificationError
alignLearningScores	learningScores, slideTime- line	slideLearningMap	None

Table 11: Exported Access Programs for Correlation and Visual Analysis Module

15.4 Semantics

15.4.1 State Variables

None.

15.4.2 Environment Variables

None.

15.4.3 Assumptions

Live video frames are synchronized with gaze streams. Slide identifiers referenced in learning scores are valid.

15.4.4 Access Routine Semantics

`mapCoordinatesToScene(worldViewData, sceneModel):`

- transition: Projects gaze vectors into pixel coordinate space.
- output: `pixelMappedData`
- exception: `MappingError`

`detectSlideTransitions(liveVideoFrames):`

- transition: Identifies slide boundaries from incoming video frames.
- output: `slideTimeline`
- exception: `VideoError`

`classifyGazeTarget(pixelMappedData, slideObjects, instructorRegions):`

- transition: Assigns semantic labels to gaze points.
- output: `labeledGazeData`
- exception: `ClassificationError`

`alignLearningScores(learningScores, slideTimeline):`

- transition: Associates learning metrics with slide intervals.
- output: `slideLearningMap`
- exception: `None`

15.4.5 Local Functions

`projectPixel: GazeVector \rightarrow PixelCoordinate`

`matchSlideRegion: PixelCoordinate \rightarrow SlideRegion`

16 MIS of Dashboard Visualization Module

16.1 Module

This module renders real-time visualization of incoming gaze and video streams for live monitoring. It displays device video feeds and derived analytics overlays when available.

16.2 Uses

Real-Time Streaming Module, Correlation & Visual Analysis Module, Engagement Analytics Module, Access Control Module, Observability Module

16.3 Syntax

16.3.1 Exported Constants

None.

16.3.2 Exported Access Programs

Name	Input	Output	Exceptions
renderPlayer	liveFrameStream: Stream-Data	uiHandle	RenderError
updateDeviceView	deviceId: DeviceID, frame: VideoFrame	status	None
displayObjectStats	labeledGazeData	statsPanel	None
showLearningScores	slideLearningMap	learningPanel	None

Table 12: Exported Access Programs for Dashboard Visualization Module

16.4 Semantics

16.4.1 State Variables

None.

16.4.2 Environment Variables

Web rendering framework and browser graphics context.

16.4.3 Assumptions

Incoming data streams are synchronized and valid.

16.4.4 Access Routine Semantics

`renderPlayer(liveFrameStream: StreamData):`

- `transition`: Initializes live rendering pipeline.
- `output`: `uiHandle`
- `exception`: `RenderError`

`updateDeviceView(deviceID: DeviceID, frame: VideoFrame):`

- `transition`: Updates displayed frame for the specified device.
- `output`: `status := updated`
- `exception`: `None`

`displayObjectStats(labeledGazeData):`

- `transition`: Updates object-level attention statistics display.
- `output`: `statsPanel`
- `exception`: `None`

`showLearningScores(slideLearningMap):`

- `transition`: Displays slide-level learning metrics.
- `output`: `learningPanel`
- `exception`: `None`

16.4.5 Local Functions

`renderFrame: VideoFrame → DOMElement`

`updateOverlay: LabeledData → OverlayLayer`

17 MIS of Reporting Module

17.1 Module

This module generates automated or customized reports integrating object-level attention, instructor-region attention, slide timelines, and learning outcome scores. Reports summarize both attention patterns and learning gains, enabling instructors to evaluate instructional effectiveness.

17.2 Uses

Dashboard Visualization Module, Engagement Analytics Module, Correlation & Visual Analysis Module, Secure Storage & Retention Module, Access Control Module

17.3 Syntax

17.3.1 Exported Constants

None.

17.3.2 Exported Access Programs

Name	Input	Output	Exceptions
generateAutoReportanalyticsData,	slide-LearningMap	reportDocument	ReportError
customizeReport	selectionCriteria, analyticsData, slideLearningMap	customReport	None
exportReport	reportDocument, format	fileOutput	FileError

Table 13: Exported Access Programs for Reporting Module

17.4 Semantics

17.4.1 State Variables

None.

17.4.2 Environment Variables

Access to file storage or cloud-based export locations.

17.4.3 Assumptions

All analytics and learning score data inputs are precomputed and validated.

17.4.4 Access Routine Semantics

`generateAutoReport(analyticsData, slideLearningMap):`

- `transition`: Produces a complete report containing aggregated attention metrics, slide timelines, and learning outcome scores.
- `output`: `reportDocument`
- `exception`: Raises `ReportError` if report generation fails.

`customizeReport(selectionCriteria, analyticsData, slideLearningMap):`

- `transition`: Generates a report restricted to selected slides, objects, or time intervals.
- `output`: `customReport`
- `exception`: None.

`exportReport(reportDocument, format):`

- `transition`: Converts the report into the specified output format and writes it to storage.
- `output`: `fileOutput`
- `exception`: Raises `FileError` if export fails.

17.4.5 Local Functions

`plotLearningVsAttention`: `AnalyticsData` \rightarrow `Figure`

`compileSlideSections`: `SlideLearningMap` \rightarrow `ReportSection`

`exportFile`: `ReportDocument` \times `Format` \rightarrow `File`

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

Stanley: Once we settled on the module list, filling in the MIS felt pretty natural and I could lean on the SRS and MG a lot.

Manan: The team was able to work well together and we were able to split the work effectively.

Angela: We were able to meet with the supervisors this week and get a good grasp on what they're looking for in terms of the designing of our modules, dashboard, and eventually, POC.

Ann: During this deliverable we got to have another meeting with our supervisors, as well as meet the PhD students who developed SocialEyes. We got to learn more about the SocialEyes framework and see how the different modules are used.

Ibrahim: Breaking the project into modules gave us a good understanding of what our next steps should be, and helped us plan ahead.

2. What pain points did you experience during this deliverable, and how did you resolve them?

Stanley: I struggled a bit with how detailed each interface should be, but looking at past MIS examples helped me find the right level.

Manan: The main pain point we experienced was coordinating with our supervisors and coming up with a list of modules we thought worked well with our system.

Angela: While we were able to derive a sufficient list of modules for our system, it did take a lot of back-and-forth, as well as brainstorming to come up with the modules, the purpose for them, etc.

Ann: Given the amount of functional and non-functional requirements listed in the SRS, I had to filter the amount that would be mapped to our modules. It was difficult

for me to come up with the top and most appropriate requirements to choose in the traceability matrix.

Ibrahim: Determining how to break the project into modules was difficult at first, as we were unsure of the exact expectations of the supervisors for each stream of the project we were planning to work on.

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

Stanley: Most of my decisions came from our meetings with the supervisors, especially their focus on making the tool genuinely useful for instructors.

Manan: Decisions regarding the modules were made after speaking with our TA and supervisor to ensure that we were on the right track. We were suggested to add more modules to better break down the system and also focus on the processing aspect of the system.

Angela: We were able to speak with both supervisors, as well as another PhD student who has experience with attention, to infer our design decisions. It really helped as well that the supervisors are not stranger's to teaching, so they're sort of the primary audience we'd be creating our modules for.

Ann: Our design decisions for the modules were inspired from our meetings with our supervisors as well as looking at the current implementation of modules and the design of the already existing SocialEyes framework.

Ibrahim: The decision to include modules for the Real-Time system and the Engagement section were made following a meeting with the supervisors.

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), if any, needed to be changed, and why?

Stanley: Nothing major needed changing, but the MIS did show a few spots in the SRS where we could clarify real-time versus post-session features later.

Manan: Nothing significantly changed for over previous documents as we had a solid foundation from our SRS and other documents.

Angela: There isn't really anything that needs to be changed stemming just from the design doc. However, that will likely change based on feedback from supervisors, TA, and peer reviews.

Ann: Nothing as of yet needed to be changed.

Ibrahim: There were no significant changes required.

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

Stanley: We're limited by hardware access and data, so with more resources I'd want more devices, more classrooms, and a more polished, customizable dashboard.

Manan: We are limited by the availability of eye-tracking devices and the accuracy of those devices. Being able to have these devices on hand at all times would make it easier to plan a system that works well with the hardware.

Angela: We don't have free reign access over using the eye-tracking devices, so we'll have to plan with the supervisors accordingly to use them.

Ann: There are multiple stretch goals and stretch functional requirements as listed in the SRS that we would love to tackle if we were given more time (privacy module, scalability, etc.)

Ibrahim: The number of eye-tracking devices available, as well the logistical challenges of setting up the lectures limits the amount of data we can test on.

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)

Stanley: We briefly considered a more monolithic design, but the current modular breakdown felt cleaner and easier to extend without rewriting everything.

Manan: We considered breaking down the modules further but we felt that the current breakdown was sufficient to cover all aspects of the system without overcomplicating it. The tradeoff with having too many modules is that it can make the system harder to manage and understand.

Angela: We briefly considered both a more monolithic design and a more fine-grained module breakdown, but chose the documented design because it provides clearer information hiding and better aligns with the SRS without adding unnecessary complexity.

Ann: We considered further breaking down our current module design smaller, but were worried about adding more complexity to the project. We wanted to carefully choose our modules for best mapping to our requirements as defined in the SRS.

Ibrahim: We considered separating some of the modules into versions for post-session and real-time analysis, but ultimately decided that the underlying frameworks would be similar and scalable enough to not have to work on them completely separately.