Module Interface Specification for Software Engineering

Team 21, Visionaries
Angela Zeng
Ann Shi
Ibrahim Sahi
Manan Sharma
Stanley Chen

January 21, 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

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

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 (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

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 identifie tracking device
SessionID	String	Unique identifie ing session
PseudonymID	String	Pseudonymous DeviceID
DataBlob	ByteArray	Raw binary dat
StreamConfig	Tuple(host: String, port: $\mathbb{N}$, bufferSize: $\mathbb{N}$)	Configuration fo
StreamSession	Tuple(sessionID: SessionID, lastFrame: VideoFrame, timestamp: $\mathbb{R}$)	Active session s
VideoFrame	ByteArray	Single frame of
SystemMetrics	Tuple(cpuUsage: $\mathbb{R}$, memoryUsage: $\mathbb{R}$, activeStreams: $\mathbb{N}$)	Runtime perfor
SystemEvent	Tuple(eventType: String, timestamp: $\mathbb{R}$, details: String)	Logged system
RetentionPolicy	Tuple(maxAgeDays: $\mathbb{N}$, maxSessions: $\mathbb{N}$)	Rules for sessio
StorageRef	String	File path refer data
StreamData	Sequence(VideoFrame)	Sequence of vid

5 Module Decomposition

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

Level 1	**Level 2**
Hardware-Hiding	Pupil Labs Neon Glasses, Central Camera, Instructor Laptop/Desktop
Behaviour-Hiding	Data Ingestion Module Real-Time Streaming Module Data Preprocessing Module Privacy Filtering Module Access Control Module Secure Storage & Retention Module Observability Module Engagement Analytics Module Correlation & Visual Analysis Module Dashboard Visualization Module Reporting Module
Software Decision	Data Storage and API Layer Authentication and Session Management Visualization Frameworks and Libraries

Table 1: Module Hierarchy for Visionaries System

6 MIS of Data Ingestion Module

6.1 Module

This module is responsible for collecting eye-tracking and video data from multiple eye-tracking devices and also the central camera. It makes sure that the raw data streams are captured, synchronized, and stored in a structured format. Making it suitable for further processing. It also collects instructor audio recordings, slide content from the lecture, and pre-/post-lecture questionnaire responses.

6.2 Uses

Real-Time Streaming Module, Data Preprocessing Module, Privacy Filtering Module, Secure Storage & Retention Module

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	Input	Output	Exceptions
collectData	deviceID, streamConfig	rawData	ConnectionError
collectAudio	deviceID	audioData	ConnectionError
collectSlides	slideSource	slideData	IOError
collectQuestionnaire	formInput	questionnaireData	FormatError
syncTimestamps	dataStreams	syncedData	SyncError
storeRawData	syncedData	status	IOError

Table 2: Exported Access Programs for Data Ingestion Module

6.4 Semantics

6.4.1 State Variables

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

6.4.2 Environment Variables

Connection to Pupil Labs Neon devices, the central camera feed, audio recording devices, slide file sources, and questionnaire input sources.

6.4.3 Assumptions

The eye-tracking devices are properly connected and the network latency is within acceptable limits for real-time data transfer.

6.4.4 Access Routine Semantics

collectData():

- transition: Starts capturing data streams from connected devices.
- output: Returns the raw data collected.
- exception: Raises ConnectionError if device is unavailable.

collectAudio():

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

collectSlides():

- transition: Retrieves slide files or slide metadata from the specified source.
- output: Returns slide data.
- exception: Raises IOError if slide files cannot be accessed.

collectQuestionnaire():

- transition: Collects pre- or post-lecture questionnaire responses.
- output: Returns questionnaire data.
- exception: Raises FormatError for incomplete or malformed questionnaire submissions.

syncTimestamps():

- transition: Aligns timestamps between multiple devices using NTP or local clock sync.
- output: Returns synchronized data streams.

- exception: Raises SyncError if timestamp alignment fails.

storeRawData():

- transition: Saves synchronized gaze, video, audio, slide, and questionnaire data to secure storage.
- output: Returns success status.
- exception: Raises IOError if data cannot be written.

6.4.5 Local Functions

validateStream(deviceID), checkIntegrity(dataChunk)

7 MIS of Real-Time Streaming Module

7.1 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.

7.2 Uses

Data Ingestion, Data Preprocessing, Privacy Filtering, Access Control, Observability

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 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 3: Exported Access Programs for Real-Time Streaming Module

7.4 Semantics

7.4.1 State Variables

activeSessions: Map(DeviceID $\rightarrow$ StreamSession)
Stores the active streaming session information for each connected device.

7.4.2 Environment Variables

Network socket interface for dashboard communication.

7.4.3 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.

7.4.4 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

7.4.5 Local Functions

validateFrame: VideoFrame $\rightarrow$ Boolean
openSocket: StreamConfig $\rightarrow$ NetworkHandle

8 MIS of Data Preprocessing Module

8.1 Module

This module prepares the captured data for analysis by filtering noise, normalizing the coordinates, and structuring the gaze and video data into useable formats. It also cleans questionnaire responses by removing empty, duplicate, or invalid entries before analysis.

8.2 Uses

Data Ingestion Module, Real-Time Streaming Module, Privacy Filtering Module

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
filterNoise	rawData	cleanData	DataError
normalizeCoordinates	cleanData	normalizedData	MathError
formatForAnalysis	normalizedData	structuredData	FormatError

Table 4: Exported Access Programs for Data Preprocessing Module

8.4 Semantics

8.4.1 State Variables

None. Operates statelessly on provided input data.

8.4.2 Environment Variables

None.

8.4.3 Assumptions

Input data follows the expected format and has valid timestamps.

8.4.4 Access Routine Semantics

filterNoise():

- transition: Applies smoothing or low-pass filters to reduce jitter.
- output: Returns cleaned gaze and video data.
- exception: Raises DataError for corrupted or incomplete data.

normalizeCoordinates():

- transition: Converts device-specific gaze coordinates to a shared reference frame.
- output: Returns normalized gaze data.
- exception: Raises MathError if coordinate mapping fails.

formatForAnalysis():

- transition: Structures data into a consistent schema for downstream analytics.
- output: Returns data in the standardized format.
- exception: Raises FormatError for invalid schema generation.

cleanQuestionnaireResponses():

- transition: Removes empty responses, malformed entries, and invalid questionnaire submissions.
- output: Returns cleaned questionnaire data ready for engagement analytics.
- exception: Raises DataError if response data cannot be validated.

8.4.5 Local Functions

interpolateMissingFrames(), alignStreams(), cleanQuestionnaireResponses()

9 MIS of Privacy Filtering Module

9.1 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. Participant consent is assumed to be handled externally prior to system execution.

9.2 Uses

Data Ingestion Module, Data Preprocessing Module

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
generatePseudonym	deviceID: DeviceID	pseudonymID: PseudonymID	PrivacyError
anonymizeParticipant	deviceID: DeviceID	anonymizedID: PseudonymID	PrivacyError

Table 5: Exported Access Programs for Privacy Filtering Module

9.4 Semantics

9.4.1 State Variables

pseudonymMap: Map(DeviceID $\rightarrow$ PseudonymID)
Stores the pseudonymous identifier associated with each device identifier for the current execution context.

9.4.2 Environment Variables

None.

9.4.3 Assumptions

Participant consent is verified externally prior to system execution.
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 or consent enforcement in code.

9.4.4 Access Routine Semantics

generatePseudonym(deviceID: DeviceID):

- transition: If deviceID $\notin$ dom(pseudonymMap), then pseudonymMap[deviceID] := newPseudonym. Otherwise no state change.
- output: pseudonymID := pseudonymMap[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: anonymizedID := pseudonymMap[deviceID]
- exception: Raises PrivacyError if the deviceID cannot be mapped to a pseudonym.

9.4.5 Local Functions

generateSecureID: DeviceID $\rightarrow$ PseudonymID
Semantics: Produces a pseudonymous identifier based on the provided DeviceID using a deterministic rule.

10 MIS of Access Control Module

10.1 Module

This module performs authorization checks and records security-relevant access events. It does not implement consent collection in code; consent is assumed to be handled externally. This module is responsible for maintaining an audit-style access log for permission checks and explicit access events.

10.2 Uses

Privacy Filtering Module

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

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

Table 6: Exported Access Programs for Access Control Module

10.4 Semantics

10.4.1 State Variables

accessLog: Sequence(AccessEvent)
Records access attempts and access-related actions.

10.4.2 Environment Variables

None.

10.4.3 Assumptions

Roles are provided by the calling context and are already authenticated outside this module. Authorization policy is simplified to role and resource checks; more granular RBAC rules may be introduced later.

10.4.4 Access Routine Semantics

validateAccess(userRole: Role, resource: ResourceID):

- transition: accessLog := accessLog $\cup$ {createAccessRecord(userRole, resource)}
- output: permissionStatus := (userRole is permitted to access resource)
- exception: Raises AuthError if the userRole is invalid or the resource identifier is malformed.

logAccess(event: AccessEvent):

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

10.4.5 Local Functions

createAccessRecord: (Role, ResourceID) $\rightarrow$ AccessEvent
Semantics: Creates an AccessEvent describing an attempted access by the given role to the given resource.

11 MIS of Secure Storage & Retention Module

11.1 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.

11.2 Uses

Data Ingestion Module, Data Preprocessing Module, Privacy Filtering Module

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	Input	Output	Exceptions
storeSessionData	sessionID: SessionID, data: DataBlob, retain: Boolean	status: Status	IOError
retrieveSessionData	sessionID: SessionID	data: DataBlob	IOError
deleteSessionData	sessionID: SessionID	status: Status	IOError
enforceRetentionPolicy	policy: RetentionPolicy	status: Status	None

Table 7: Exported Access Programs for Secure Storage & Retention Module

11.4 Semantics

11.4.1 State Variables

storedSessions: Map(SessionID $\rightarrow$ StorageRef)
Maps each stored session to a storage reference (for example, a file path).

11.4.2 Environment Variables

Local file system access for reading and writing session artifacts.

11.4.3 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.

11.4.4 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

11.4.5 Local Functions

resolveStorageRef: SessionID $\rightarrow$ StorageRef
Semantics: Determines the storage reference used for a given session.

12 MIS of Observability Module

12.1 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.

12.2 Uses

Real-Time Streaming Module, Secure Storage & Retention Module, Access Control Module

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

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

Table 8: Exported Access Programs for Observability Module

12.4 Semantics

12.4.1 State Variables

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

12.4.2 Environment Variables

System monitoring interfaces available to the runtime environment.

12.4.3 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.

12.4.4 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

12.4.5 Local Functions

summarizeMetrics: Sequence(SystemMetrics) $\rightarrow$ SystemReport
Semantics: Produces a SystemReport summarizing recent system metrics.

13 MIS of Engagement Analytics Module

13.1 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.

13.2 Uses

Data Preprocessing Module, Privacy Filtering Module

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
computeLearningScores	anonymizedQuestionnaireData, slideMap	learningScores	DataError

Table 9: Exported Access Programs for Engagement Analytics Module

13.4 Semantics

13.4.1 State Variables

None.

13.4.2 Environment Variables

None.

13.4.3 Assumptions

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

13.4.4 Access Routine Semantics

computeLearningScores():

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

13.4.5 Local Functions

computeLearningDelta(), aggregateSlideScores()

14 MIS of Correlation and Visual Analysis Module

14.1 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.

14.2 Uses

Data Preprocessing Module, Engagement Analytics Module

14.3 Syntax

14.3.1 Exported Constants

None.

14.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
mapCoordinatesToScene	worldViewData, sceneModel	pixelMappedData	MappingError
detectSlideTransitions	liveVideoFrames	slideTimeline	VideoError
classifyGazeTarget	pixelMappedData, slideObjects, instructorRegions	labeledGazeData	ClassificationError
alignLearningScores	learningScores, slideTimeline	slideLearningMap	None

Table 10: Exported Access Programs for Correlation and Visual Analysis Module

14.4 Semantics

14.4.1 State Variables

None.

14.4.2 Environment Variables

None.

14.4.3 Assumptions

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

14.4.4 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

14.4.5 Local Functions

projectPixel: GazeVector $\rightarrow$ PixelCoordinate
matchSlideRegion: PixelCoordinate $\rightarrow$ SlideRegion

15 MIS of Dashboard Visualization Module

15.1 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.

15.2 Uses

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

15.3 Syntax

15.3.1 Exported Constants

None.

15.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
renderPlayer	liveFrameStream: StreamData	uiHandle	RenderError
updateDeviceView	deviceID: DeviceID, frame: VideoFrame	status	None
displayObjectStats	labeledGazeData	statsPanel	None
showLearningScores	slideLearningMap	learningPanel	None

Table 11: Exported Access Programs for Dashboard Visualization Module

15.4 Semantics

15.4.1 State Variables

None.

15.4.2 Environment Variables

Web rendering framework and browser graphics context.

15.4.3 Assumptions

Incoming data streams are synchronized and valid.

15.4.4 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

15.4.5 Local Functions

renderFrame: VideoFrame $\rightarrow$ DOMElement
updateOverlay: LabeledData $\rightarrow$ OverlayLayer

16 MIS of Reporting Module

16.1 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.

16.2 Uses

Engagement Analytics Module, Dashboard Visualization Module

16.3 Syntax

16.3.1 Exported Constants

None.

16.3.2 Exported Access Programs

Name	Input	Output	Exceptions
generateAutoReport	analyticsData, slideLearningMap	reportDocument	ReportError
customizeReport	selectionCriteria, analyticsData, slideLearningMap	customReport	None
exportReport	reportDocument, format	fileOutput	FileError

Table 12: Exported Access Programs for Reporting Module

16.4 Semantics

16.4.1 State Variables

None.

16.4.2 Environment Variables

Access to file storage or cloud-based export locations.

16.4.3 Assumptions

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

16.4.4 Access Routine Semantics

generateAutoReport():

- transition: Produces a full report with aggregated attention graphs, instructor-region attention, slide-object attention, and learning outcome scores.
- output: Returns a comprehensive report document.
- exception: Raises ReportError if report generation fails.

customizeReport():

- transition: Creates a report for selected slides, objects, audio portions, or instructor-region attention metrics.
- output: Returns a customized report document.
- exception: None.

exportReport():

- transition: Converts the report to the specified format (PDF, HTML, etc.) and writes it to storage.
- output: Returns the exported file.
- exception: Raises FileError if export fails.

16.4.5 Local Functions

plotLearningVsAttention(), compileSlideSections(), exportFile()

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 though worked well with out 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), it 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 chooose 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 similar and scaleable enough to not have to work on them completely separately.