Module Interface Specification for Room8

Team 19
Mohammed Abed
Maged Armanios
Jinal Kasturiarachchi
Jane Klavir
Harshil Patel

January 17, 2025

1 Revision History

Date	Version	Members	Notes
2025-01-17	0.0	Team	Initial creation

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/jinalkast/room8/blob/main/docs/SRS/SRS.pdf

Contents

3 Introduction

The following document details the Module Interface Specifications (MIS) for Room8, a suite of tools for providing roommates in shared living situations tools to settle common disputes and keep track of shared responsibilities. Beginning from and following Section 7 the MIS for all modules in the Module Guide (MG) can be found. Potential readers of this document include new developers, and maintainers looking for clarity on the implementation of modules.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/jinalkast/room8/tree/main.

4 Module Decomposition

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

Level 1	**Level 2**
Hardware-Hiding Module	M1 M2 M3
Behaviour-Hiding Module	M4 M7 M8
Software Decision Module	M5 M6 M9 M10 M11 M12 M13 M14

Table 1: Module Hierarchy

5 MIS of Sensor Reading Module

5.1 Module

M1: SensorReadingModule

5.2 Uses

Used to gather data on user presence in shared space to determine when to take before and after pictures of the shared space.

5.3 Syntax

5.3.1 Exported Constants

None

5.3.2 Exported Access Programs

Name	In	Out	Exceptions
timeMotion	timesOfMotion	timeMotionDetected	-
detector	sensorData	motionPresent	-

5.4 Semantics

5.4.1 State Variables

- timesOfMotion: List[datetime] - List of time when motion was detected by the sensor.

5.4.2 Environment Variables

- sensorData: sensorDataType - Data acquired by the sensor with data type supported by sensor.

5.4.3 Assumptions

None

5.4.4 Access Routine Semantics

timeMotion(timesOfMotion: datetime) → datetime:

- input: List of time stamps of when motion was detected.
- output: Last timestamp of when motion was detected.

detector(sensorData: sensorDataType) $\rightarrow$ boolean:

- input: Data received from sensor.
- output: True or false value depending on if motion was detected.

5.4.5 Local Functions

- insertTime(): Inserting time stamp into timesOfMotion variable.

6 MIS of Image Capture Module

6.1 Module

M2: ImageCaptureModule

6.2 Uses

Capture image of shared space using camera system.

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
captureImage	-	image	-

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

- captureImage will only be called when picture is needed to be taken, that is before user starts using shared space and five minutes after user has left shared space.

6.4.4 Access Routine Semantics

captureImage() $\rightarrow$ png:

- output: Image taken.

6.4.5 Local Functions

None

7 MIS of Image Upload Module

7.1 Module

M3: ImageUploadModule

7.2 Uses

Uploads the captured image to Raspberry Pi for the cleanliness detection system to use for analysis.

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
uploadImage	image	-	imgUploadErrorMessage

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

- uploadURL: str - The URL to which the image will be uploaded.

7.4.3 Assumptions

- Network is stable and never causes error in image upload.
- Power source is stable and never causes error in image upload.

7.4.4 Access Routine Semantics

uploadImage(image: png):

- input: Image taken from camera.
- exception: ImageUploadInterrupted - Raised if error occurs during image upload.

7.4.5 Local Functions

None

8 MIS of Image Preprocessing Module

8.1 Module

M4: ImagePreprocessingModule

8.2 Uses

The ImagePreprocessingModule is used for preparing raw images (i.e. from the ImageUploadModule) to be submitted to subsequent modules (i.e. ObjectDetectionModule and ScoringModule). A series of transformations is applied to the image.

8.3 Syntax

8.3.1 Exported Constants

- SUPPORTED_FORMATS: List[str] = [”JPEG”, ”PNG”]
- DEFAULT_TRANSFORMS: torchvision.transforms.Compose A default set of PyTorch transformations, including resizing, normalization, and optional filtering

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
processImage	image: Image	Tensor	-

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

None

8.4.3 Assumptions

- The input image is in a valid format supported by PyTorch (e.g. JPEG, PNG)
- PyTorch and torchvision libraries are installed and functional
- Network connectivity is stable for receiving images through the cloud network

8.4.4 Access Routine Semantics

processImage(image: Image):

- transition: Applies a sequence of PyTorch transformations to the input image
- output: Returns the transformed image as a Pytorch tensor, ready for subsequent object detection
- exception:
  - InvalidImageFormatException: Raised if the input image format is unsupported
  - TransformationError: Raised if an error occurs during transformations

processImage(image: Image):

- transition: Applies pixel map transformations (e.g. resizing, cropping, filtering) to the input image
- output: Returns the transformed image
- exception:
  - TransformationError: Raised if an error occurs during transformations

8.4.5 Local Functions

- validateImageFormat(image: Image) $\rightarrow$ bool: Checks if the input image format is supported
- createTransforms() $\rightarrow$ torchvision.transforms.Compose Creates a PyTorch Compose object for the default set of transformations
- applyTransforms(image: Image, transforms: torchvision.transforms.Compose) $\rightarrow$ Tensor: Applies the given transformations to the input image

9 MIS of Object Detection Module

9.1 Module

M5: ObjectDetectionModule

9.2 Uses

To detect objects in an input tensor (preprocessed image) using a pretrained object detector (e.g. Faster R-CNN ResNet-50 FPN model). The output is a list of detected objects, where each object is represented as a HouseObject.

9.3 Syntax

9.3.1 Exported Constants

- MODEL_NAME: str = ”FasterRCNNResNet50FPN”
- CONFIDENCE_THRESHOLD: float = 0.5 (minimum confidence score for object detection results)

9.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
detectObjects	inputTensor: Tensor	List[HouseObject]	-

9.4 Semantics

9.4.1 State Variables

- self.model: torchvision.models.detection.FasterRCNN

9.4.2 Environment Variables

None

9.4.3 Assumptions

- The input tensor is correctly preprocessed and normalized according to the requirements of the Faster R-CNN model
- PyTorch and torchvision libraries are installed and functional

9.4.4 Access Routine Semantics

detectObjects():

- transition: Uses the pretrained Faster R-CNN model to detect objects in the input tensor, filters results based on the confidence threshold
- output: Returns a list of detected objects, where each object is represented as a House-Object
- exception:
  - ModelNotLoadedException: Raised if the model fails to load
  - DetectionError: Raised if an error occurs during detection process

9.4.5 Local Functions

- loadModel() → torchvision.models.detection.FasterRCNN: Loads pretrained Faster R-CNN ResNet-50 FPN model
- filterDetections(detections: List[Dict], threshold: float) → List[HouseObject]: Filters the raw detections based on the confidence threshold and converts them to HouseObject instances

10 MIS of Scoring Module

10.1 Module

M6: ScoringModule

10.2 Uses

To evaluate changes in a room by comparing two sets of detected objects (representing ”before” and ”after” states) and assign a cleanliness score based on the differences

10.3 Syntax

10.3.1 Exported Constants

- MAX_SCORE: int = 100 (maximum cleanliness score)
- MIN_SCORE: int = 0 (minimum cleanliness score)
- OBJECT_WEIGHTS: dict (Weights for scoring different objects based on their categories)

10.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
calculateCleanliness	before: List[HouseObject], after: List[HouseObject]	int	-

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

None

10.4.3 Assumptions

- Input lists ’before’ and ’after’ contain valid HouseObject instances
- The HouseObject class provides sufficient information (e.g., label, confidence, bounding box) to compare objects between the two states

10.4.4 Access Routine Semantics

calculateCleanliness():

- transition: Compares the before and after lists to identify added, removed, or moved objects; computes a cleanliness score based on these changes
- output: Returns an integer cleanliness score between MIN_SCORE and MAX_SCORE
- exception:
  - ModelNotLoadedException: Raised if the model fails to load
  - DetectionError: ScoringError: Raised if an error occurs during the scoring process

10.4.5 Local Functions

- compareObjects(before: List[HouseObject], after: List[HouseObject]) → Dict[str, List[HouseObject]] Identifies added, removed, and moved objects between the two lists
- computeScore(changes: Dict[str, List[HouseObject]]) → int: Computes the cleanliness score based on detected changes

11 MIS of Request Listener Module

11.1 Module

M7: RequestListenerModule

11.2 Uses

Exposes cleanliness detector to camera and image by making it an application programming interface.

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
getImages	-	images	-

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

None

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

getImages() $\rightarrow$ List[png]:

- output: Two most recent images where one is the before and other is the after state of shared space after user is finished.

11.4.5 Local Functions

None

12 MIS of Data Uploading Module

12.1 Module

M8: DataUploadingModule

12.2 Uses

To provide an interface for the app's backend to interact with the Python-based object detection, machine learning, and scoring functionalities. The module packages these functionalities using FastAPI and exposes them as RESTful endpoints.

12.3 Syntax

12.3.1 Exported Constants

- API_VERSION: str = "v1"
- BASE_URL: str = "/api/v1"

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
startServer	-	None	-
POST /processImages	imageSet: JSON	cleanlinessScore: JSON	-

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

- FastAPI: Required for exposing RESTful endpoints
- Backend: The JavaScript backend that interacts with the module

12.4.3 Assumptions

- FastAPI and its dependencies (e.g., Uvicorn) are installed and properly configured
- The backend sends valid JSON requests with correctly encoded image data

12.4.4 Access Routine Semantics

startServer():

- transition: Initiates the FastAPI application and starts the server to handle incoming requests
- output: None
- exception: ServerError - Raised if the server fails to start

Endpoint: POST /processImages

- transition:
  - Decodes the base64-encoded input images
  - Passes the images through the ObjectDetectionModule and ScoringModule
  - Computes and returns the cleanliness score
- output: A JSON object with the following structure:
  {
  "cleanliness_score": int
  }
- exception:
  - InvalidInputError: Raised if the input JSON is malformed or invalid
  - ProcessingError: Raised if an error occurs during processing

12.4.5 Local Functions

- decodeImages(encodedImages: List[str]) $\rightarrow$ List[Tensor]: Decodes base64-encoded images into PyTorch tensors
- processRequest(images: List[Tensor]) $\rightarrow$ int: Processes the input images through the ObjectDetection and Scoring modules to compute the cleanliness score

13 MIS of Chore Scheduling Module

13.1 Module

M9: ChoreSchedulingModule

13.2 Uses

The Chore Scheduling Module is used to assist roommates in scheduling chores. It provides a list of chores to be done and allows roommates to assign chores to each other. The module is in charge of schedule's database table, management of schedule instances, the application programming interface of the scheduling functions, and the implenentation to the frontend application.

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
getHouseSchedule	string: houseID	JSON: houseScheduleObject	JSON: errorObject

13.4 Semantics

13.4.1 State Variables

- houseSchedule: JSON - A JSON object containing the current house's schedule
- selectedRoommate: string - The roommate selected to view schedule

13.4.2 Environment Variables

- SUPABASE_API_KEY: Key required to interface with the Supabase backend, which stores bill-related data.

13.4.3 Assumptions

- The houseScheduleObject is a valid JSON object that corresponds to the house's schedule

13.4.4 Access Routine Semantics

getHouseSchedule(string: houseID) → JSON:

- output: Returns the house schedule object for the specified houseID containing the schedule for all roommates
- exception: Raises an exception if the houseID is invalid or if the house schedule object is not found

13.4.5 Local Functions

- getRoommateHouseSchedule(JSON: houseScheduleObject, string: roommateID) → JSON: Returns the schedule for the specified roommate
- updateChore(string: houseID, string: choreID, JSON: newChore) → JSON: Updates the house schedule with the new chore for the specified previous chore
- createChore(string: houseID, JSON: newChore): Creates a new house chore object inside of the house schedule
- deleteChore(string: houseID, string: choreID) → JSON: Deletes the specified chore from the house schedule

14 MIS of Chat Bot Module

14.1 Module

M10: ChatBotModule

14.2 Uses

The ChatBot Module interfaces with an SMS API to facilitate group messaging functionality. It allows users to view the current group SMS status for the house, create a group, and send messages. The module implements logic for sending relevant notifications.

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
createGroup	groupName: str, members: List[str]	GroupID: str	InvalidGroupException
sendMessage	groupID: str, message: str	bool	MessageDeliveryException

14.4 Semantics

14.4.1 Environment Variables

- SMS_API_KEY: key required to interface with the API which sends the SMS messages to the group.
- SUPABASE_API_KEY: Key required to interface with the Supabase backend, which stores bill-related data.

14.4.2 Assumptions

- The SMS API is operational and has sufficient credits for sending messages.
- The provided phone numbers of all users in the house/group are valid.

14.4.3 Access Routine Semantics

createGroup(groupName: str, members: List[str]):

- transition: Adds a new group to ‘activeGroups‘ with the specified members.
- output: Returns a unique group ID for the newly created group.
- exception: InvalidGroupException - Raised if the group exceeds the maximum allowed size or has invalid member details.

sendMessage(groupID: str, message: str):

- transition: Sends the message to all members of the specified group via the SMS API.
- output: Returns ‘true‘ if the message is successfully sent, otherwise ‘false‘.
- exception: MessageDeliveryException - Raised if the message fails to deliver due to API issues or invalid group ID.

14.4.4 Local Functions

- validatePhoneNumber(phone: str) $\rightarrow$ bool: Ensures the phone number follows the correct format.
- generateGroupID(groupName: str) $\rightarrow$ str: Generates a unique group ID based on the group name and timestamp.
- sendViaSms(phone: str, message: str) $\rightarrow$ bool: Sends a message to a specified phone number using the SMS API.

15 MIS of Bill Splitting Module

15.1 Module

M11: BillSplittingModule

15.2 Uses

The Bill Splitting Module calculates the division of bills among users and assigns outstanding debts/credits to individual users. It allows users to create, modify, and delete existing bill data from the client-side interface.

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
createBill	billDetails: dict	BillID: str	InvalidBillException
modifyBill	billID: str, updatedDetails: dict	bool	BillNotFoundException, InvalidBillUpdateException
deleteBill	billID: str	bool	BillNotFoundException
calculateSplit	billID: str	SplitDetails: dict	CalculationErrorException

15.4 Semantics

15.4.1 Environment Variables

- SUPABASE_API_KEY: Key required to interface with the Supabase backend, which stores bill-related data.

15.4.2 Assumptions

- All user data (e.g., names, amounts owed) is valid and stored in the Supabase backend.
- Users provide accurate and consistent data for creating or modifying bills.
- The Supabase backend is operational and responsive.

15.4.3 Access Routine Semantics

createBill(billDetails: dict):

- transition: Sends the provided bill details to the Supabase backend for storage.
- output: Returns a unique Bill ID for the newly created bill.
- exception: InvalidBillException - Raised if the provided bill details are incomplete or invalid.

modifyBill(billID: str, updatedDetails: dict):

- transition: Updates the details of an existing bill in the Supabase backend.
- output: Returns 'true' if the modification is successful.
- exception:
  - BillNotFoundException - Raised if the specified Bill ID does not exist.
  - InvalidBillUpdateException - Raised if the updated details are invalid or conflict with existing data.

deleteBill(billID: str):

- transition: Deletes the specified bill from the Supabase backend.
- output: Returns 'true' if the deletion is successful.
- exception: BillNotFoundException - Raised if the specified Bill ID does not exist.

calculateSplit(billID: str):

- transition: Retrieves bill details from the Supabase backend and calculates the split among participants.
- output: Returns a dictionary containing split details (e.g., each participant's share, amounts owed/credited).
- exception: CalculationErrorException - Raised if an error occurs during the calculation (e.g., missing participant data).

15.4.4 Local Functions

- validateBillDetails(details: dict) $\rightarrow$ bool: Ensures the provided bill details (e.g., amounts, participants) are valid.
- calculateIndividualShare(amount: float, participants: List[str]) $\rightarrow$ float: Divides the bill amount evenly among participants.
- updateDebtCredit(participantID: str, amount: float) $\rightarrow$ bool: Updates the outstanding debt/credit for a participant in the Supabase backend.
- removeBill(billID: str) $\rightarrow$ bool: Deletes the bill record from the Supabase backend.

16 MIS of User Authentication Module

16.1 Module

M12: UserAuthenticationModule

16.2 Uses

The User Authentication Module is used to authenticate users and manage user accounts. It provides a secure way for users to log in and access the application. The module is in charge of the user's database table, management of user instances, the application programming interface of the authentication functions, and the implementation to the frontend application.

16.3 Syntax

16.3.1 Exported Constants

- SUPPORTED_AUTHENTICATION_METHODS: List[str] = ["OAuth2"]
- SUPPORTED_AUTHENTICATION_PROVIDERS: List[str] = ["Google"]

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
getUser	string: jwtCookie	JSON: userObject	-

16.4 Semantics

16.4.1 State Variables

- activeJWT: datetime - The current accepted web token timing for user authentication

16.4.2 Environment Variables

- SUPABASE_API_KEY: Key required to interface with the Supabase backend, which stores bill-related data.

16.4.3 Assumptions

- Google authentication service is always available and not down.
- The user has a valid Google email account for authentication.

16.4.4 Access Routine Semantics

getUser(jwtCookie: string) → JSON:

- transition: Validates the JWT cookie and retrieves the corresponding user data from the UserDatabase.
- output: Returns the userObject containing user details.
- exception:
  - InvalidJWTException: Raised if the JWT cookie is invalid or expired.
  - UserNotFoundException: Raised if no user is found for the given JWT cookie.

16.4.5 Local Functions

- validateJWT(jwt: string) → bool: Validates the given JWT token and returns true if valid, false otherwise.
- generateJWT(userID: string) → string: Generates a new JWT token for the given user ID.
- signInWithGAuth(authCode: string) → JSON: Authenticates the user using Google OAuth and returns user details.
- deleteJWT(jwt: string) → bool: Deletes the given JWT token and returns true if successful, false otherwise.

17 MIS of Home Management Module

17.1 Module

M13: HomeManagementModule

17.2 Uses

The Home Management Module is used to manage house-related operations such as adding, deleting, editing, joining, and leaving a house. It provides a way for users to manage their house and roommates within the application. The module is in charge of the house's database table, management of house instances, the application programming interface of the house management functions, and the implementation to the frontend application.

17.3 Syntax

17.3.1 Exported Access Programs

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
getRoommates	string: houseID	JSON: roommateList	JSON: errorObject
getHouseID	string: userID	string: houseID	JSON: errorObject
getHouseMetadata	string: houseID	JSON: houseMetadata	JSON: errorObject

17.4 Semantics

17.4.1 State Variables

- houseList: List[JSON] - The list of houses managed by the module
- roommateList: List[JSON] - The list of roommates for the specified house

17.4.2 Environment Variables

- SUPABASE_API_KEY: Key required to interface with the Supabase backend, which stores bill-related data.

17.4.3 Assumptions

None

17.4.4 Access Routine Semantics

getRoommates(houseID: string) → JSON:

- output: Returns the list of roommates for the specified houseID.
- exception: HouseNotFoundError: Raised if the houseID does not exist.

getHouseID(userID: string) → string:

- output: Returns the houseID that the user with the specified userID belongs to.
- exception: UserNotFoundError: Raised if the userID does not exist.

getHouseMetadata(houseID: string) → JSON:

- output: Returns the metadata for the specified houseID.
- exception: HouseNotFoundError: Raised if the houseID does not exist.

17.4.5 Local Functions

- addHouse(houseData: JSON) → JSON: Adds a new house to the HouseDatabase and returns the created house object.
- deleteHouse(houseID: string) → JSON: Deletes the specified house from the HouseDatabase and returns a confirmation message.
- editHouse(houseID: string, houseData: JSON) → JSON: Edits the details of the specified house in the HouseDatabase and returns the updated house object.
- joinHouse(userID: string, houseID: string) → JSON: Adds the user to the specified house and returns a confirmation message.
- leaveHouse(userID: string, houseID: string) → JSON: Removes the user from the specified house and returns a confirmation message.
- checkHouseExists(houseID: string) → bool: Checks if a house with the specified houseID exists in the HouseDatabase.
- checkUserExists(userID: string) → bool: Checks if a user with the specified userID exists in the UserDatabase.
- updateHouseList(): Updates the houseList state variable with the latest data from the HouseDatabase.

18 MIS of Cleanliness Management Module

18.1 Module

M14: CleanlinessManagementModule

18.2 Uses

The Cleanliness Management Module converts data received from the camera system and scoring module into a user-friendly interface. It provides users with the ability to view the details of messes in their shared space, including an image of the mess, the assigned user responsible for cleaning, and a toggle to mark the mess as cleaned.

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
getMessDetails	messID: str	MessDetails: dict	MessNotFoundException
markAsCleaned	messID: str, userID: str	bool	AuthorizationException, MessNotFoundException
assignMessToUser	messID: str, userID: str	bool	MessNotFoundException, UserNotFoundException

18.4 Semantics

18.4.1 Environment Variables

- SUPABASE_API_KEY: Key required to interface with the Supabase backend, which stores bill-related data.

18.4.2 Assumptions

- The camera system and scoring module provide accurate and timely data.
- All users have unique identifiers and are registered in the Supabase backend.
- Users are responsible for marking messes as cleaned only if they are assigned to them.
- The Supabase backend is operational and responsive.

18.4.3 Access Routine Semantics

getMessDetails(messID: str):

- transition: Retrieves details of the specified mess from the Supabase backend.
- output: Returns a dictionary containing mess details, including the image, assigned user, and cleanup status.
- exception: MessNotFoundException - Raised if the specified mess ID does not exist in the backend.

markAsCleaned(messID: str, userID: str):

- transition: Updates the status of the specified mess to ”cleaned” in the Supabase backend.
- output: Returns ‘true‘ if the operation is successful.
- exception:
  - AuthorizationException - Raised if the user attempting to mark the mess as cleaned is not assigned to it.
  - MessNotFoundException - Raised if the specified mess ID does not exist.

assignMessToUser(messID: str, userID: str):

- transition: Updates the assigned user for the specified mess in the Supabase backend.
- output: Returns ‘true‘ if the operation is successful.
- exception:
  - MessNotFoundException - Raised if the specified mess ID does not exist.
  - UserNotFoundException - Raised if the specified user ID does not exist in the backend.

18.4.4 Local Functions

- fetchCameraData(messID: str) → dict: Retrieves image data and metadata for the specified mess from the camera system.
- updateMessStatus(messID: str, status: str) → bool: Updates the cleanup status of the mess in the Supabase backend.
- assignUserToMess(messID: str, userID: str) → bool: Assigns the specified user to the mess in the Supabase backend.
- validateUserPermission(userID: str, messID: str) → bool: Ensures the user is authorized to modify the mess’s status.

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?**
   When writing this deliverable, the team addressed and resolved questions about our project that we had been putting off for a long time. Decisions such as "how *exactly* will we be implementing this" were answered, and now the team is fully aligned on how each module will interact. Another thing that went well when writing this deliverable was that, unlike previous deliverables, there were no conflicts or issues raised because of past deliverables. This is because the team went back and modified past deliverables to check that all deliverables were in line with each other and wouldn't cause obstacles in the future.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   Some pain points when writing this deliverable included a lack of clarity about specific solution details and uncertainty about the granularity of modules. When writing the MG and the MIS, the team was still unclear about the exact outputs of the cleanliness detection algorithm. While working on the document, the team was unsure if we would implement advanced functionality like detecting stains and which user created them or if we would fall back to more basic functionality. Additionally, the team was unclear if we should include modules not implemented by us completely, such as authentication frameworks, databases, web frameworks, and programming languages. The lack of clarity about the cleanliness detection algorithm, while not completely resolved, was handled by specifying our unknowns in the anticipated changes section of our module guide and creating clear pathways for development based on what happens in the future. Additionally, we decided not to pursue one of our ideas due to privacy and other concerns raised by our project supervisor. The clarity issues about the document itself were rectified by communicating with our TA, including more modules, and distinguishing modules that were not implemented but used in the MG.

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?**
   Almost all decisions stemmed from speaking to client(s) or a proxy. Our proxy for design decisions related to the cleanliness management system and all related systems is Dr. Tharmarasa, an expert in computer vision and object detection. Decisions related to the features of the client-facing application come from discussions with students in shared-living situations which are abundant at McMaster University.

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?**
   The SRS was modified to remove requirements that involved recording the individual cleanliness score of a user.

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)
   Current limitations in the solution include limitations in the detection ability of the cleanliness detection algorithm. As of writing, this module is implemented using pre-trained object detection models and focuses on detecting differences in the state, leaving the users to determine whether or not the state of the room has improved or worsened. This is due to a lack of time and expertise in implementing a custom AI model for this use case. Given unlimited time and resources, a dedicated artificial intelligence and object detection model specializing in detecting ”messes” would be built to reduce manual effort from the user and improve the system's accuracy.

   Another limitation of the system is its inability to enforce accountability amongst roommates. Currently, the solution provides a suite of tools for roommates to use and to keep track of information but does not provide any way of enforcing things like bill payments or chores. Given more time, the team could have implemented some point or merit tracking system that would report the landlord who isn't paying bills and keeping their property in proper order, and the landlord could provide notices to tenants and help with enforcement.

6. **Give a brief overview of other design solutions you considered. What are the benefits and trade-offs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design?** (LO_Explores)
   Some alternative solutions were the creation of a mobile app over a progressive web application (PWA), the creation of our machine learning model over a pre-trained model, and having the camera system track which user modified the shared living space and keeping track of which user contributed the most to issues in shared living spaces. The first two alternatives provide the development team with more flexibility and customizability but increase development time due to the team's lack of experience building AI models and mobile apps. Ultimately, the team decided to use a pre-trained

AI model and build a PWA because the deadlines in this project are firm, and being unable to deliver any product is worse than a product with less functionality. Finally, the team decided not to pursue implementing the solution, which involved tracking which users did what in the shared living situations due to concerns raised by our supervisor. These concerns included privacy concerns and accuracy concerns. Since the team is using a pre-trained AI model, we decided it would be best to not associate data with users to prevent frustrations caused by false positives/negatives (defined in the Hazard Analysis).