Software Requirements Specification

for

JARVIS AI AGENT

Version 1.0

2025-12-09

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Initial Draft	2025-12-09	Initial setup	V1.0.0

1. Introduction

The JARVIS AI AGENT System is a all-in-one AI solution which aims to integrate voice recognition, hand gesture recognition, and face recognition to allow full automation of a smart home. The program will also use facial biometrics and speaking detection for authorization of commands.

1.1 Purpose

The purpose of this project is to deploy home automation using efficient machine learning to run on microcomputers. It will be accessible to anyone who has camera access. The program will be deployed to a mobile phone and utilize the camera and microphone of the device to process commands.

1.2 Project Scope

The scope of the project is to ship a complete AI package that will run on a smart phone and includes image processing combined with voice recognition to allow the user to fully automate their smart home using gesture commands and microphones. The first version of this project will run on a laptop camera and microphone for PoC testing, second prototype will be integrated with a database to allow the storing and modification of commands across multiple devices. The third prototype will be deployed to a smartphone to allow the user to utilize a mobile device.

1.3 References

1.3.1 Research Links

- OpenCV startup guide for C++
- OpenCV
- Executing C++ from NodeJS using child_process (Fallback method)
- Executing C++ from JS using WebAssembly (Primary method)
- Challenges of gesture tracking with computer vision
- Gesture capture with computer vision approaches
- Example work in python for gesture capture
- Example work in python using OpenCV for gesture capture
- ASL hand gestures for computer vision training
- Computer vision entire hand tracking

1.3.2 Definitions

- Media Pipe Training Set

1.3.3 Abbreviations

2. Overall Description

Smartphone integration to an AI app that contains hand gesture and voice recognition using facial biometrics for ID.
The anticipated user is a more tech literate user who understands junior level ideas.
The assumption is that the user has an smartphone device and internet connection.
The dependency for this project is a static hosted Linux server.

2.1 Product Perspective

This is an entirely new product designed from the ground up.

2.2 User Classes and Characteristics

Service Moderator: Technical support workers for the product.
Characteristics: High technical skills, strong understanding of application systems.

All below will belong to one or many “Families” or “Groups”

Account Admin: Highest end user permission level of the application.
Characteristics: Owner of the consumer product, moderate technical skills

Account User: Moderate end user permission level of application.
Characteristics: Generated by account admin, low technical skills, cannot add new commands

Account Guest: Low end user permission level of application.
Characteristics: Generated by account admin, low or no technical skills, cannot access the mobile app.

2.3 Operating Environment

The system will operate on a mobile smartphone with the primary processing of images, facial biometrics, and voice features being offloaded to a dedicated Linux server. Because this system can be handled via a smartphone the system can be operated from anywhere with internet connectivity.

The server which is running this prototype is using a MySQL database running multiple other services, along with other APIs

2.4 Design and Implementation Constraints

Because the main processing suite will be in a C language, it will be more difficult to allow local processing of image, and audio data. Because of this the data will be uploaded to the server on a set interval while the app is open and in use and the server will process the data and return a response with its decisions.

2.4.1 Stack

The computer vision model will be written in CPP using the OpenCV framework
The main computer vision model will leverage the Media Pipe framework for hand tracking training data
The API will be handled using an expressJS API running on the dedicated server.
The database will be a MySQL database and will run on the dedicated server.
The frontend UI will operate on a smartphone with under 2gb of memory usage and minimal CPU usage to prevent battery wastage.

2.4.2 Test Stack

All CPP applications will be tested using GTest
All JS applications will be tested using Jest

2.5 Assumptions and Dependencies

Assumptions:
- User owns a smartphone
- User smartphones are connected to the internet

Dependencies:
- Dedicated host server

3. System Features

3.1 User facial biometrics for authentication

3.1.1 Description

The user will be able to use their face to authenticate commands and have gestures using facial biometric data. This feature is considered a medium priority for the development of the application. Once an authenticated user is present but has gone out of view of the camera, the authentication status will remain for 5 minutes.

3.1.2 Stimulus/Response Sequences

- User(s) will enter the frame of the camera.
- Every few frames the AI will reconsider the image and identify all faces present.
- If one or more faces are identified the AI will consider it against the user list associated with that group
- If one or more faces comes back as registered. The person with the highest permission level will be used for

the authentication of commands.

- The smartphone app will create a popup saying "Welcome (Name of highest ranked user)"

3.1.3 Functional Requirements

- Uploading biometric data to the facial recognition algorithm.
- Storing biometric data for retrieval later.
- Camera capture.
- A working facial detection algorithm.

3.2 User voice recognition for commands

3.2.1 Description

Users will be able to vocally speak while the app status is authenticated. The speech the user performs will be converted into text and returned as a sequence of zero to many predefined commands. This feature is considered highly important for the completion of this app

3.2.2 Stimulus/Response Sequences

- User performs authentication routine.
- User begins to speak
- Mobile app begins to convert the audio to UTF-8 text
- Mobile app waits for a 1 second pause in the user's speech
- Mobile app submits the text to the API server
- API server extracts commands from the text
- API begins to execute the commands, and returns the list of commands registered back to the mobile application
- Mobile application displays the commands that are being executed while the commands are being executed from the server.
- A transcription of the event is stored containing who said the commands, what commands were run, what time, who was present at the time of execution, and the name of the command.

3.2.3 Functional Requirements

- Speech recognition suite running on server instance
- API capable of extracting predefined list of commands from human speech running on server
- Response time of API to be less than 1 second
- Storage database capable of storing the transaction
- Storage database containing the commands
- API capable of parsing and executing the extracted commands

3.3 User hand gestures for commands

3.3.1 Description

Users will be able to create static hand gestures with their hand once authenticated. Once the user is authenticated the mobile app will capture the camera every few frames and upload it to the API. The API will look for a predefined list of hand gestures and attempt to identify them. If the API is confident in its results, it will parse the command from the gesture and execute the command. This feature is considered highly important for the completion of this app.

3.3.2 Stimulus/Response Sequences

- User performs authentication routine.
- User remains within the frame of the camera and provides a static hand gesture
- The frame is uploaded to the API and parsed, attempting to identify the hand gesture against a set of predefined ones
- If a hand gesture is identified confidently it is sent to the API to be executed as a command query
- The command query is executed, and sent back to the mobile app
- The mobile app displays what command is being executed
- A transcription of the event is stored containing who gestured the commands, what commands were run, what time, who was present at the time of execution, and the name of the command.

3.3.3 Functional Requirements

- image recognition suite running on server instance
- image model capable of extracting predefined lists of commands from gestures running on server
- Response time of image model to be less than 1 second
- Storage database capable of storing the transaction
- Storage database containing commands and their corresponding gestures
- API capable of parsing and executing the extracted commands

3.4 Dynamic granular permission levels controlled by the user

3.4.1 Description

Users will be able to create a “Group” or “Family” within their house and set Admins, Users, Guests. All commands will have 2 checkboxes next to them “Users can execute”, “Guests can execute”. This allows admins to set granular permissions to all commands on the app. By default, new commands can be used by everyone. This is considered of low importance to the completion of the app.

3.4.2 Stimulus/Response Sequences

- When an admin creates a new command, it is displayed in a list along with the other commands
- The mobile view is scrolled to render the new command
- To the left of the command are two checkboxes showcasing the permissions
- Users can click on checkboxes to toggle them, and the mobile app will async send a request to the API to store this permission
- If the request comes back negative a toast will be displayed showing error and the action reverted
- If the request is successful nothing is displayed. Besides the initial change conducted before the API call.

3.4.3 Functional Requirements

- User session tokens to authenticate user interactions.
- Database storage of commands and user permissions.
- API to handle communication between these two services.

3.5 User customization of commands and gestures

3.5.1 Description

Admin users will be able to upload their own gestures and register their own commands to the system to allow them to set up the app to their own home requirements. Once commands or gestures are registered the entire “Group” or “Family” can then use those commands if they are granted permissions to their user level. This is considered of medium importance to the completion of the application.

3.5.2 Stimulus/Response Sequences

- User is signed in or authenticated and viewing the mobile app
- User enters the commands menu
- User selects the add new command
- User enters the command name, and an optional description of what it does
- User optionally selects the “Assign Gesture” button and recreates the gestures for the image model to learn from
- User hits “Save” once the command is prepared
- The view is scrolled to the bottom of the command list showing the new command

3.5.3 Functional Requirements

- Database storage of commands
- Storage of the gestures, either on the image model or in the database
- Mobile client interface for CRUD’ing the gestures
- API for cross service communication.

4. Use Cases

4.1 Creating an account

Use Case Name:	Creating an account
Scenario:	User is not authenticated and wants to sign up
Triggering Event:	User selects “Sign Up” under the login screen
Actors:	User, System, API, Database, image model
Activities:	1. New page is rendered showing the user the signup process has begun 2. Signup process asks for a name, an email, a recovery password 3. Signup process then asks for facial biometric data and asks the user to scan the full 180 front of their face 4. This data is uploaded to the database 5. This data is then used in the image model to authenticate the user based on their facial biometrics 6. The API creates a log of the event

4.2 Deleting an account

Use Case Name:	Deleting an account
Scenario:	User is authenticated and wants to delete their account and all their data
Triggering Event:	User enters their settings, hits the delete account button, presses okay on a popup warning all data regarding their user will be deleted
Actors:	User, System, API, Database
Activities:	1. User selects to delete their account 2. A popup appears warning the user that this action is not reversible and all their data will be purged 3. If the user selects okay, the request to the API to purge all data is sent 4. While the data is being purged, sign the user out and delete the local storage Cache 5. The user will be shown the login page

4.3 Updating an account

Use Case Name:	Updating an account

Scenario:	User is authenticated and wants to update their information
Triggering Event:	User selects “Update” under the settings screen
Actors:	User, System, API, Database
Activities:	1. New page is rendered showing the user account configurations 2. Allow the user to update zero to many fields: [email, password] 3. If one or more fields are modified allow the user to click save 4. When the user clicks save send the request to the API 5. Show a loading icon while the info is being processed 6. The API updates the database information 7. The API returns a status code 8. The spinner stops and renders the homepage 9. The API creates a log of the event

4.4 Seamless authentication

Use Case Name:	Seamless authentication
Scenario:	User is not signed in, they want to sign in without a password, and camera access is granted
Triggering Event:	user selects “Sign in with face” on the login page
Actors:	User, System, API, Database, image model
Activities:	1. New page is rendered telling the user to look at the camera 2. A camera preview is displayed showing what the camera is seeing in 15 fps 3. Every 3 frames, a frame is uploaded to the API 4. The API forwards the information to an image model 5. The image model attempts to link a userID to the face 6. If the image model is successful, the user is signed in 7. If the image model is unsuccessful 10 times the email + password login method is shown to the user. 8. The API creates a log of the event.

4.5 Gesture commands

Use Case Name:	Gesture Commands
Scenario:	User is authenticated and camera access is granted

Triggering Event:	User gestures a command
Actors:	User, System, API, Database, image model
Activities:	1. The camera is captured and sent to the API paired with user information 2. The image model identifies the user in the picture and the gesture 3. The API confirms the user identification belongs to the group or family of the signed in user 4. The API processes the command 5. The API sends a response to the client containing what command was executed 6. The mobile client displays a notification showing the command was executed 7. The API creates a log of the event

4.6 Voice Commands

Use Case Name:	Voice Commands
Scenario:	User is authenticated and microphone access is granted
Triggering Event:	The user says “Hey Jarvis”
Actors:	User, System, API, Database
Activities:	1. The voice capture begins to listen to speech until it detects at least 2 seconds pause in speech. 2. An API request is sent containing the full speech transcription 3. The API extracts commands from the speech against a list of predefined commands to create a list of requested commands 4. Commands are executed in order of the list 5. Once the command list is generated a response is sent to the client containing the list of commands to be executed 6. The list is displayed to the user sequentially 7. The API creates a log of the event

4.7 Creating Commands

Use Case Name:	Creating Commands
Scenario:	User wants to create new commands for the application and user is authenticated
Triggering Event:	User selects “Add New Command” button

Actors:	User, System, API, Database
Activities:	1. User is signed in or authenticated and viewing the mobile app 2. User enters the commands menu 3. User selects the add new command 4. User enters the command name, and an optional description of what it does 5. User optionally selects the "Assign Gesture" button and recreates the gestures for the image model to learn from 6. If step 5 is done and camera permission is not granted, a request to access camera is sent 7. User hits "Save" once the command is prepared 8. The view is scrolled to the bottom of the command list showing the new command 9. The API creates a log of the event

4.8 Adding guests, or users to the group or family

Use Case Name:	Adding guests, or users to the group or family
Scenario:	User wants to add family members or guest to access the application
Triggering Event:	User selects "add new user"
Actors:	User, System, API, Database
Activities:	1. A screen pops up asking for a userID 2. The user enters the userID and selects the users permission from a dropdown menu 3. The user selects save

4.9 Transaction History

Use Case Name:	Transaction History
Scenario:	User wants to see an activity history
Triggering Event:	User selects "Activity"
Actors:	User, API, Database
Activities:	1. The new screen is rendered 2. The past 10 activities are displayed 3. The user can select forwards & backwards to retrieve up until the end of

	the trips

5. Data Requirements

Server file structure

- Planning
- Image Processor
    - TrainingData
    - DataSet
        - (UserID from database)
- API
    - Routes
    - Controllers
    - DataAccessors
    - Database
    - Validators
    - Auth
    -

Mobile App file structure

- App
    - SRC
        - Views
        - Sign Up Screen
        - Homepage
        - Family View
        - Command View
    - HELPERS
    - UTILS

5.1 Reports

Transcriptions of ALL actions will be logged, categorized by the primary mode of:
userAction

commandAction
familyAction

Followed by a secondary category of:
userUpdated
userDeleted
userCreated
commandUpdated
commandDeleted
commandCreated
commandExecuted
familyUpdated
familyDeleted
familyCreated

5.2 Data Acquisition, Integrity, Retention, and Disposal

All transactions or transcripts will be stored forever with no planned automated deletion.
User data will be stored until user account deletion
User biometrics data will be stored on the local server as unencrypted .jgp files and will be deleted upon user account deletion

6. Requirements

6.1 User Interfaces

6.1.1 Views

- Login screen
- Sign up screen
    - User biometrics gathering
    - User account details entering
- Homepage
    - Creating families
    - Settings
        - Delete account
        - Update account
- Families view
- Family view
    - Modifying family users
    - Deleting the family

- Command view
    - Modifying commands
        - Gathering training material for gestures
    - Deleting commands
- User history
- Family history
- Command history

6.1.2 Error messages

6.1.2.1 User facing mobile errors

All user facing errors will be displayed as a toast notification containing “ERROR” as its first word

6.1.2.2 Dev facing mobile errors

All console logged mobile errors will be printed to console with “[ERROR]” as its first word

6.1.2.3 Dev facing server errors

All console logged server errors will be printed to console with “[ERROR]” as its first word

6.1.2.4 Dev facing database errors

All console logged database errors will be printed to console with “[ERROR]” as its first word

6.1.2.5 Dev facing computer vision errors

All console logged computer vision errors will be printed to console with “[ERROR]” as its first word

6.1.3 Network requests

All network requests will follow RESTful HTTP protocols and be sent through a dedicated middleman on the client side.

6.2 Software Interfaces

6.2.1 Mobile APP

The mobile app will only communicate to the server API via RESTful HTTP protocols

6.2.2 Server API

The server API will return responses to the mobile APP via RESTful HTTP protocols
The server API will connect to the database for reading and writing via js MySQL library
The server API will connect to the computer vision code via child processes and return values

6.2.3 Computer Vision

The computer vision will connect to the Server API as a child and return responses as program exit values

6.2.4 Database

The database will only return responses to the server API via the js MySQL library

6.3 Hardware Interfaces

There will be two hardware interfaces:

1. *Mobile app*
    a. *Iphone*
    b. *Android*
    c. *Linux OS*
2. *Dedicated server*
    a. *Ubuntu*

6.4 Communications Interfaces

Communication will occur with return values from CPP files
Communication will occur with HTTP packets using RESTful protocols for network communication
The phones operating system, touch display drivers, microphone drivers, and camera drivers will be used for user input

7. Quality Attributes

7.1 Usability

The mobile app shall be deployable to any smartphone system. But for development we will only use Android.

7.2 Performance

The smartphone must respond to gestures within 3 seconds of invoking one
The smartphone must allow UI changes that only modify the database to be fully completed and responded to within 1 second

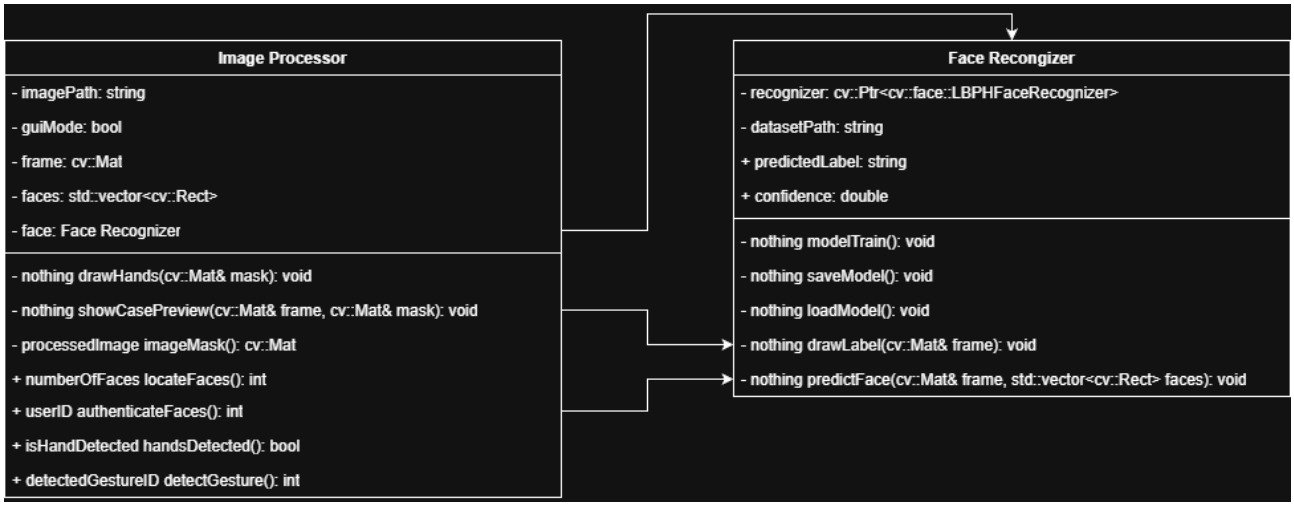
7.3 Security

Passwords will be hashed using MD5 before transit
User sessions will use JWT protocols for management.
API will have a middleware auth layer
All API endpoints will have a content validation layer

8. System Diagrams

9. Diagrams Folder

9.1 Image Model



9.2 API Endpoints

endpoint	Type	Header	Body	Return
/status	GET	NONE	NONE	{status}
/auth	POST	NONE	{email, passwordHash}	{status, sessionToken}
/user	GET	sessionToken	NONE	{name}
/user/settings	GET	sessionToken	NONE	{email, name, updated_at}
/user/families	GET	sessionToken	NONE	{familyIDs, familyNames}
/user	PATCH	sessionToken	Optional {email, name, passwordHash, biometricBlobs}	{status}
/user	DELETE	sessionToken	NONE	{status}
/user	POST	NONE	{email, name, passwordHash, biometricBlobs}	{status}
/family/all	GET	sessionToken	NONE	{familyNames, familyIDs}
/family/{ID}	GET	sessionToken	NONE	{familyName, familySize, commandSize, commandNames, commandIDs}
/family	POST	sessionToken	{familyName}	{status, familyID}
/family/{ID}	PATCH	sessionToken	{familyName}	{status}
/family/{ID}	DELETE	sessionToken	NONE	{status}
/commands	POST	sessionToken	{commandName, commandExecution, familyID}	{status, commandID}
/commands/{ID}	PUT	sessionToken	{handGestureBlobs}	{status}

/commands/{ID}	PATCH	sessionToken	Optional {commandName, commandExecution}	{status}
/commands/{ID}	DELETE	sessionToken	NONE	{status}

9.3 Database