SeeFood App

Software Design Specification and Planning

By Group 14: Nathaniel Crossman | Muhammad Khan | Don Miller | Ryan Zink

OCTOBER 26, 2017

CEG 4110

Process Specification

Scenarios

(1) Scenario:
Scenario Title: Application Startup
Actors/Roles:

- **User**: is anyone using the SeeFood application
- **Client**: android SeeFood application
- **Apache webserver:** running on EC2 instance
- **REST API**: hosted on Apache webserver

Scenario Description:

The **User** clicks the SeeFood app icon located on their local mobile device and then the **System** displays the loading screen. Furthermore, during this loading process, the **System** checks to make sure that it is connected to the Internet and can connect to the EC2 server, as well, before displaying the **System** home screen.

(2) Scenario:
Scenario Title: Analyzing Image from Camera
Actors/Roles:

- **User**: is anyone using the SeeFood application
- **Client**: android SeeFood application
- **REST API**: hosted on Apache webserver
- **AI API**: hosted on Apache webserver
- **Apache webserver:** running on EC2 instance

Scenario Description:

The **User** clicks on the camera icon button from home screen and is redirected to that camera page. The **User** takes a picture and the image is sent to the **Apache Webserver** for image Analysis. Once the analysis process is done, the resulting score gets sent back to the **Client** and displayed on the user's home screen.

(3) Scenario:
Scenario Title: Analyzing Image from Gallery
Actors/Roles:

- **User**: is anyone using the SeeFood application
- **Client**: android SeeFood application
- **REST API**: hosted on Apache webserver
- **AI API**: hosted on Apache webserver
- **Apache webserver:** running on EC2 instance

Scenario Description:

The **User** clicks on the Gallery icon button from home screen and is redirected to that gallery page. The **User** selects one or many pictures and the pictures are sent to the **Apache Webserver** for image Analysis. Once the analysis process is done, the resulting score gets sent back to the user and displayed on the user's home screen.

(4) Scenario:
Scenario Title: User Clicks Thumbnail
Actors/Roles:

- **User**: is anyone using the SeeFood application
- **Client**: android SeeFood application
- **AI API**: hosted on Apache webserver
- **REST API**: hosted on Apache webserver
- **Apache webserver:** running on EC2 instance

Scenario Description:
The **User** clicks on a thumbnail image located inside of the **System** view image page. The **System** sends an HTTP GET request to the **Apache Webserver** which is running mod_wsgi. Through mod_wsgi the **Apache Webserver** routes the request to the **Python API** which calls a flask operation /getImage [GET]. The **API** builds a JSON data package of a full-size version of the image with certain key attributes stored inside of the JSON data package. This data package is sent back to the **System** and the **System** is able to view a full-size image.

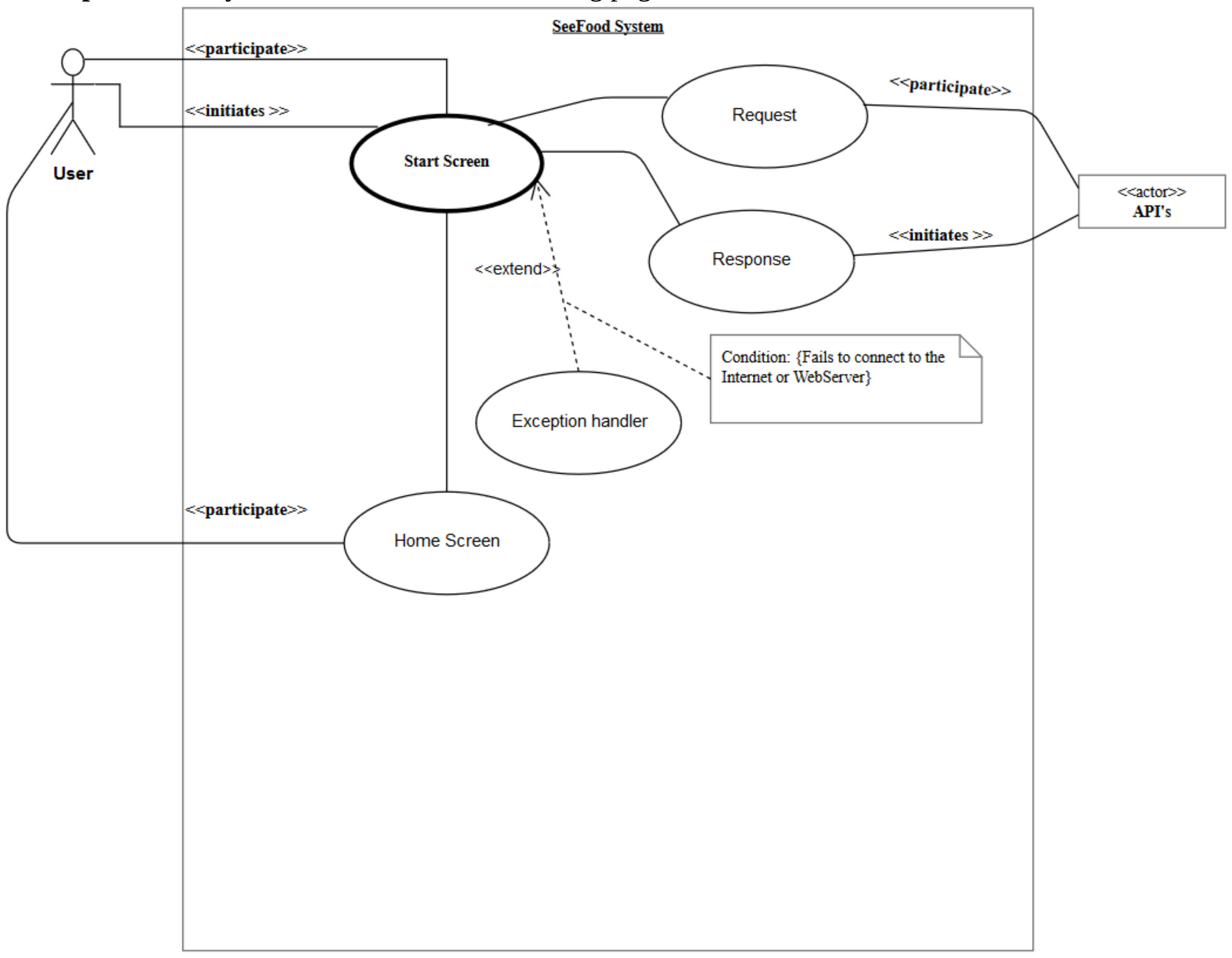
(5) Scenario:
Scenario Title: View Past Images
Actors/Roles:

- **User**: is anyone using the SeeFood application
- **Client**: android SeeFood application
- **AI API**: hosted on Apache webserver
- **REST API**: hosted on Apache webserver
- **Apache webserver:** running on EC2 instance

Scenario Description:
The **User** clicks on the thumbnail icon button which is located in the drop-down menu on the home screen and gets redirected to the view image page. The **System** sends an HTTP GET request to the **Apache Webserver** which is running mod_wsgi. Through mod_wsgi the **Apache Webserver** routes the request to the **Python API** which calls a flask operation. This operation builds a JSON data package of thumbnail images and their corresponding scores. Then this data package is sent back to the **System** which displays the thumbnails.

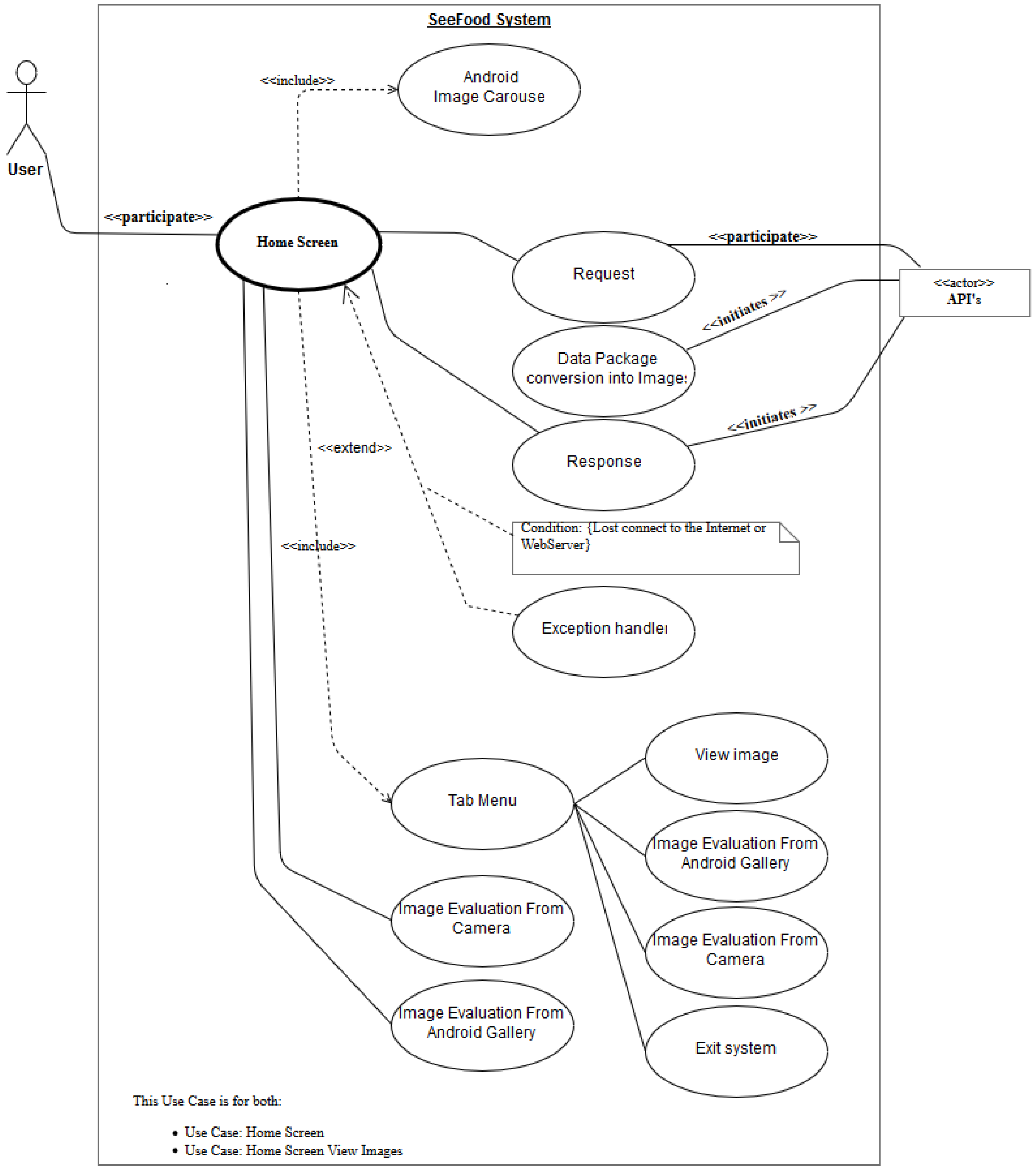
Use-cases

Use Case: Start Screen
Description: The system loads the SeeFood loading page.



Use Case: Home Screen

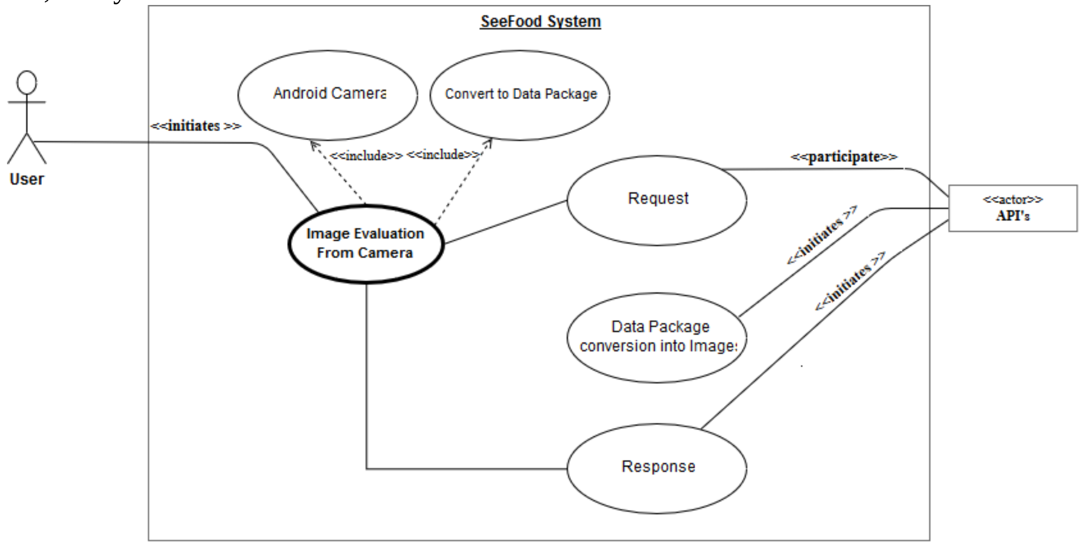
Description: The home screen is displayed. The user takes (camera) or selects (gallery) an image and sends it to the EC2 server for evaluation. The user navigates to the image view. The image view displays thumbnails and evaluations of previous images which are stored on the server.


SeeFood System
User
<<participate>>
<<include>>
Android Image Carouse
Home Screen
Request
<<participate>>
<<actor>>
API's
<<initiates >>
Data Package conversion into Image:
<<initiates >>
Response
<<extend>>
Condition: {Lost connect to the Internet or WebServer}
<<include>>
Exception handler
View image
Tab Menu
Image Evaluation From Android Gallery
Image Evaluation From Camera
Image Evaluation From Camera
Exit system
Image Evaluation From Android Gallery
This Use Case is for both:
Use Case: Home Screen
Use Case: Home Screen View Images

Use Case: Camera

Description

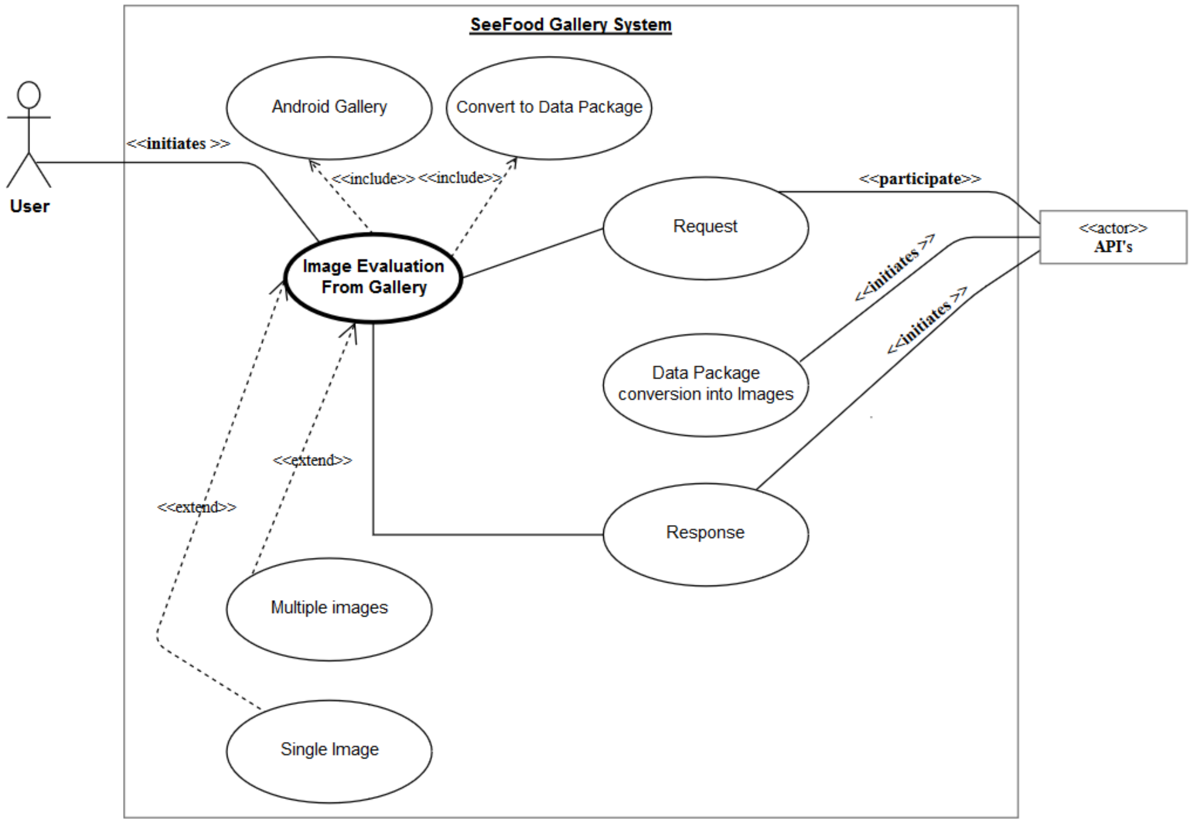
The user is takes a picture and the picture are sent to the EC2 search for valuation. Once the user has sent the picture, the system redirects the user to the home screen.



Use Case: Local Gallery

Description

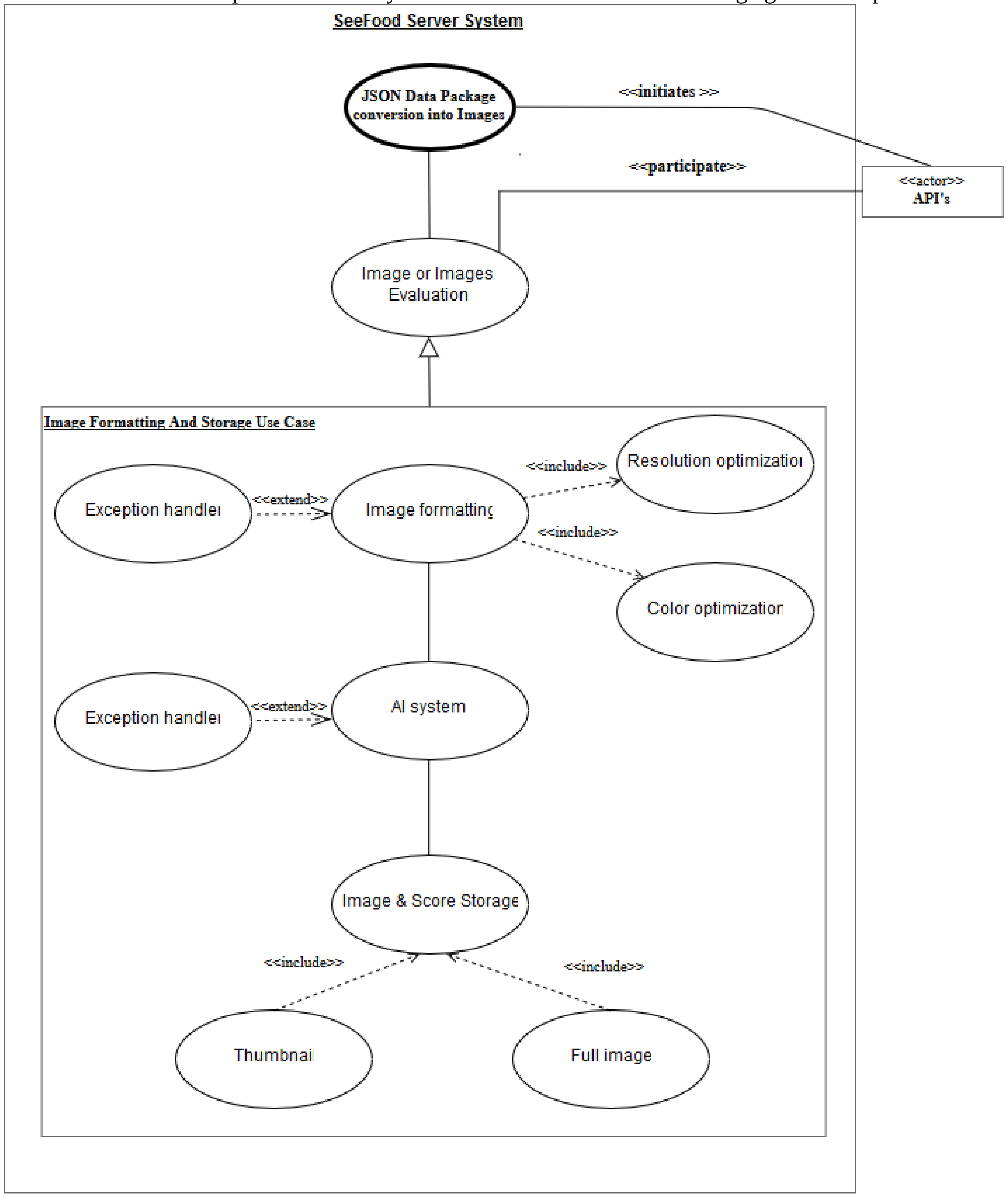
The user can select from available images stored on their local android device to be evaluated on the EC2 server. Additionally, the user can select and send more than one image for evaluation. Lastly, once images are sent to the server the user gets redirected back to the home screen.



Use Case: EC2 Server Image Evaluation

Description

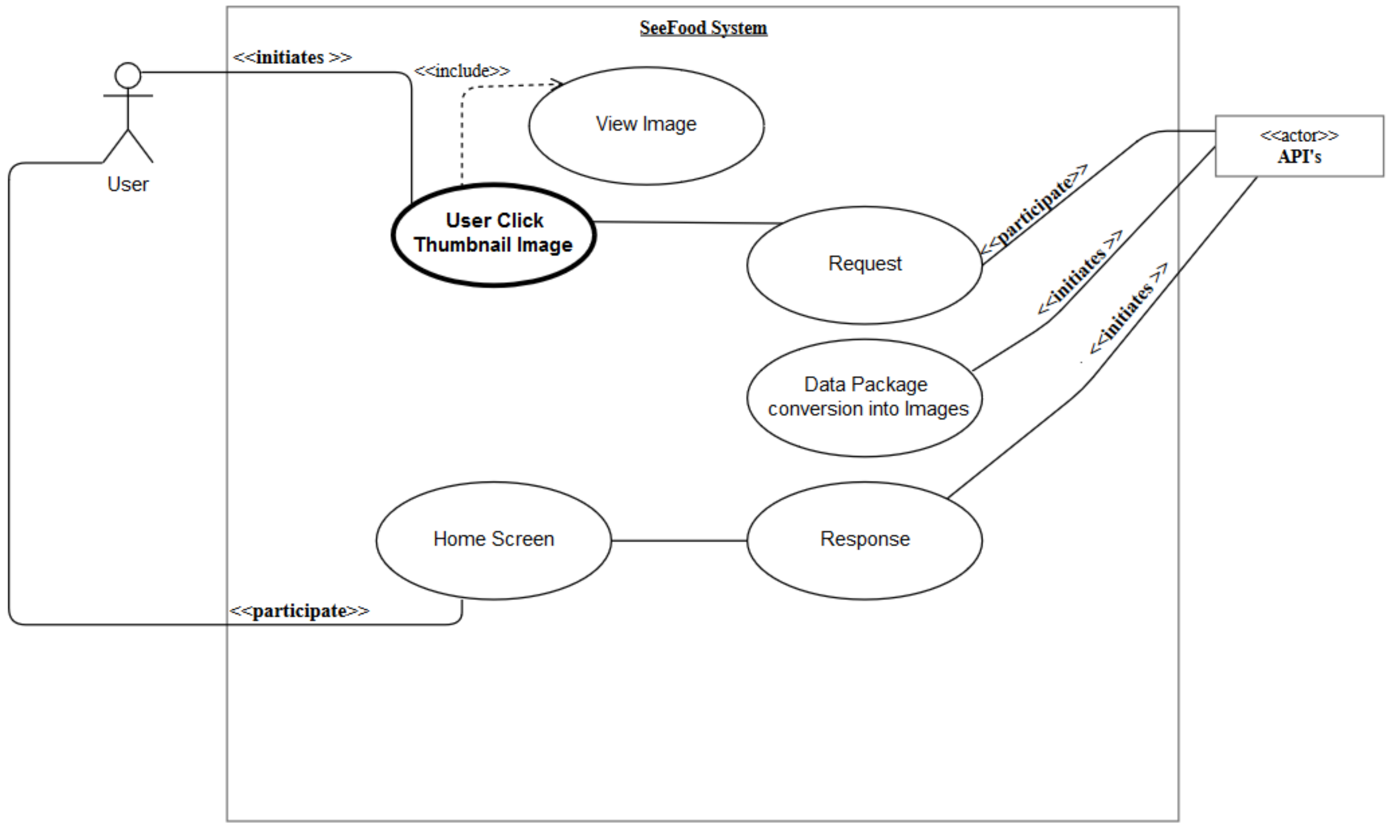
The EC2 server receives a HTTPS data package from one of its ports. This package is then handled by the API system; whereupon, it is evaluated and then passed to the AI system which evaluates the pictures or picture. Once this is completed the API system sends it back over the listening age HTTPS port.



Use Case: View Full Size Image

Description

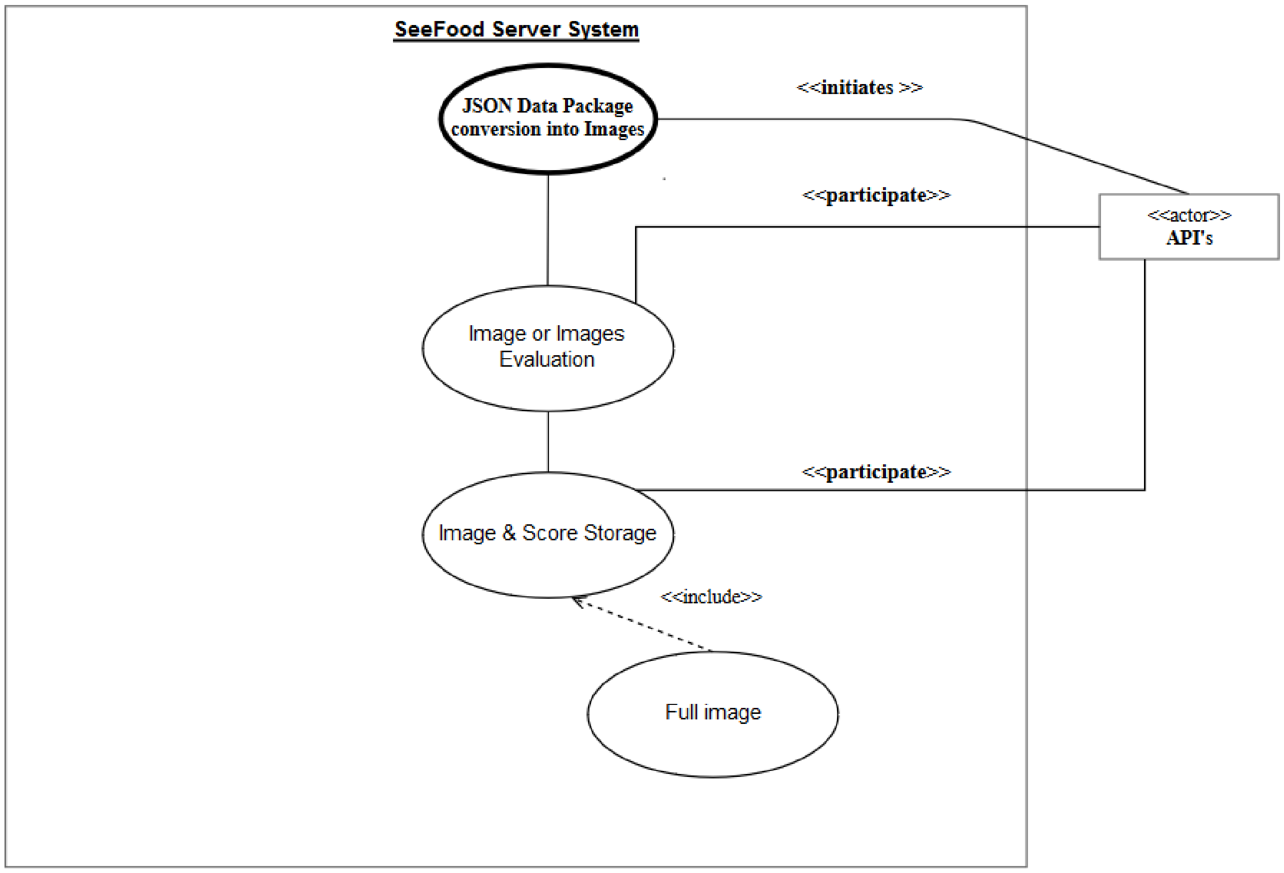
The user can view full size image from the local thumbnails gallery.



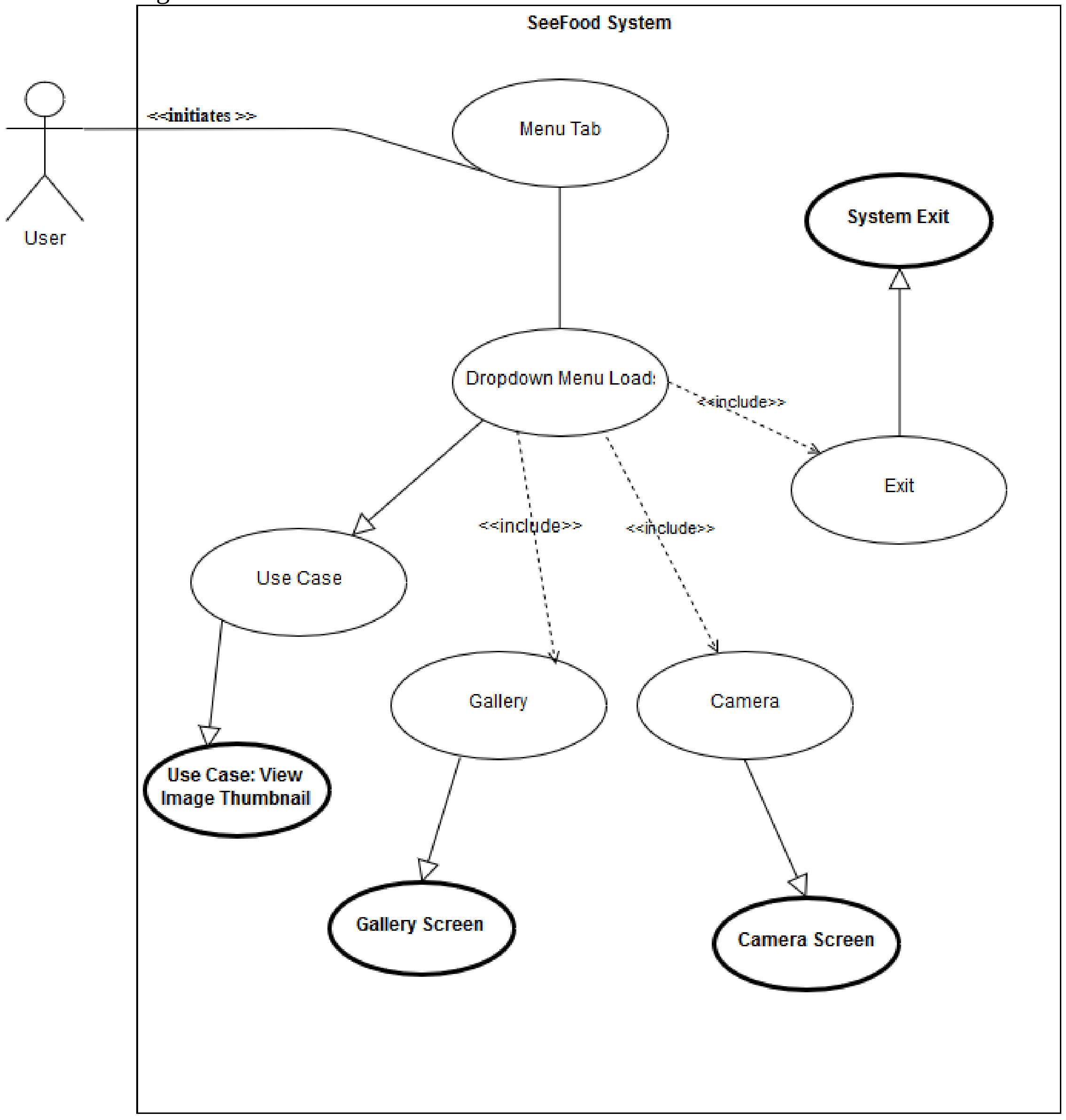
Use Case: Server Get Full Size Image

Description

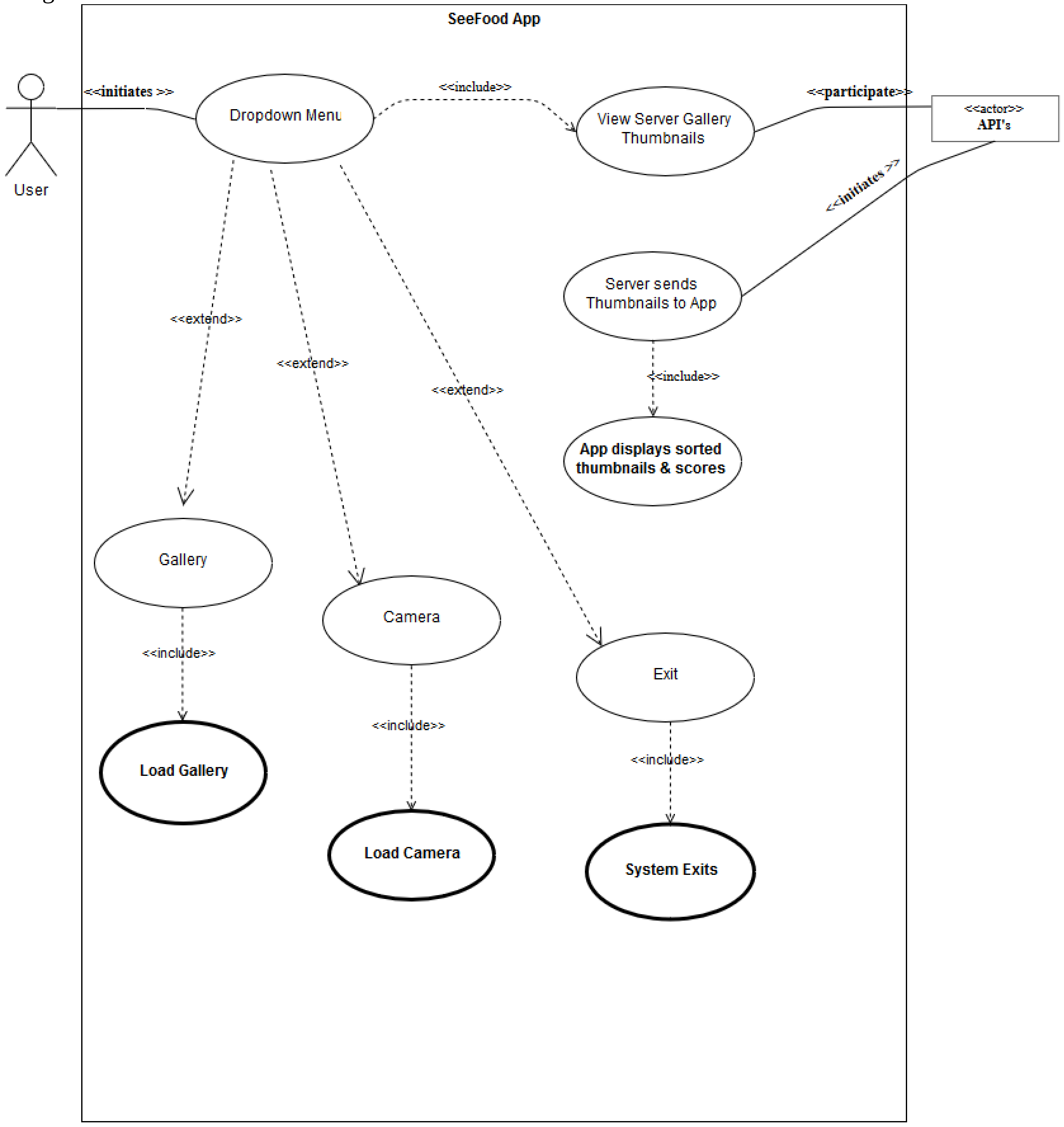
The client's request a full size image and the server response with the requested image.



Use Case: Home Screen Menu Tab
Description
The user navigates to the Menu Tab.

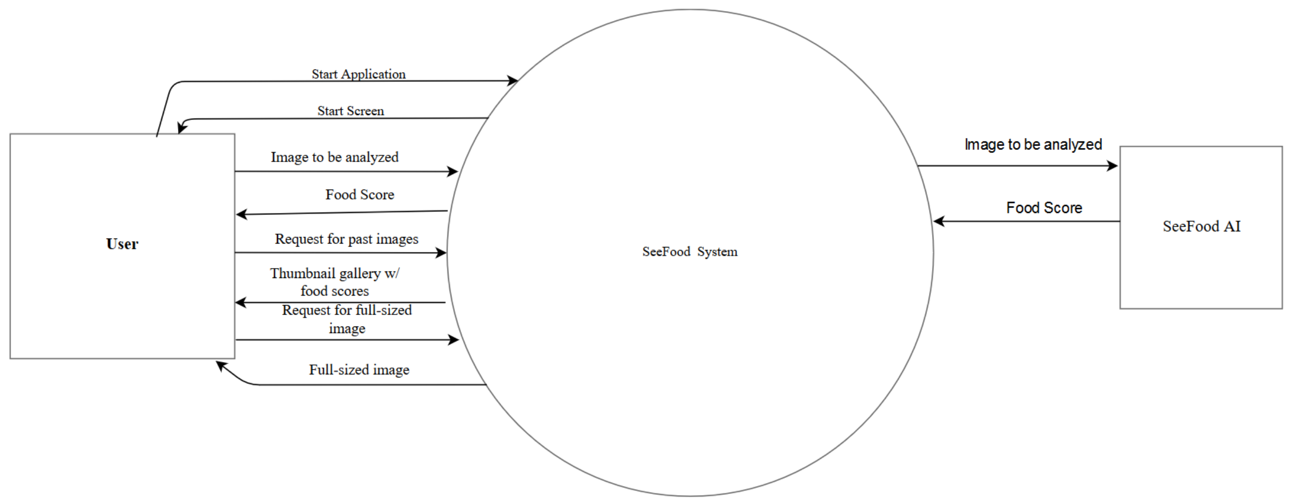


Use Case: View Image Thumbnails
Description
The user navigates to the image view. The image view displays thumbnails and evaluations of previous images which are stored on the server.

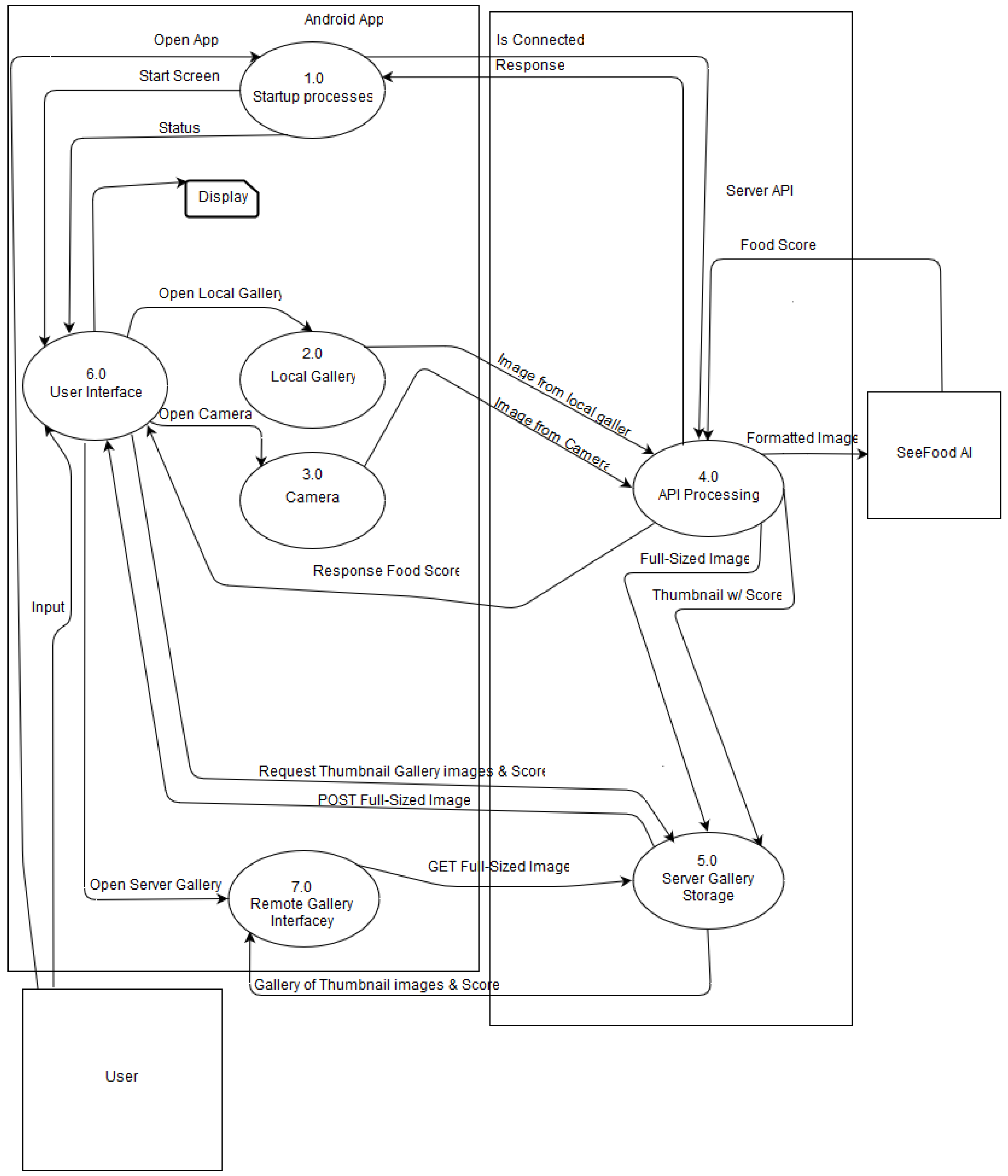


Semi-Formal Model

Top-Level DFD

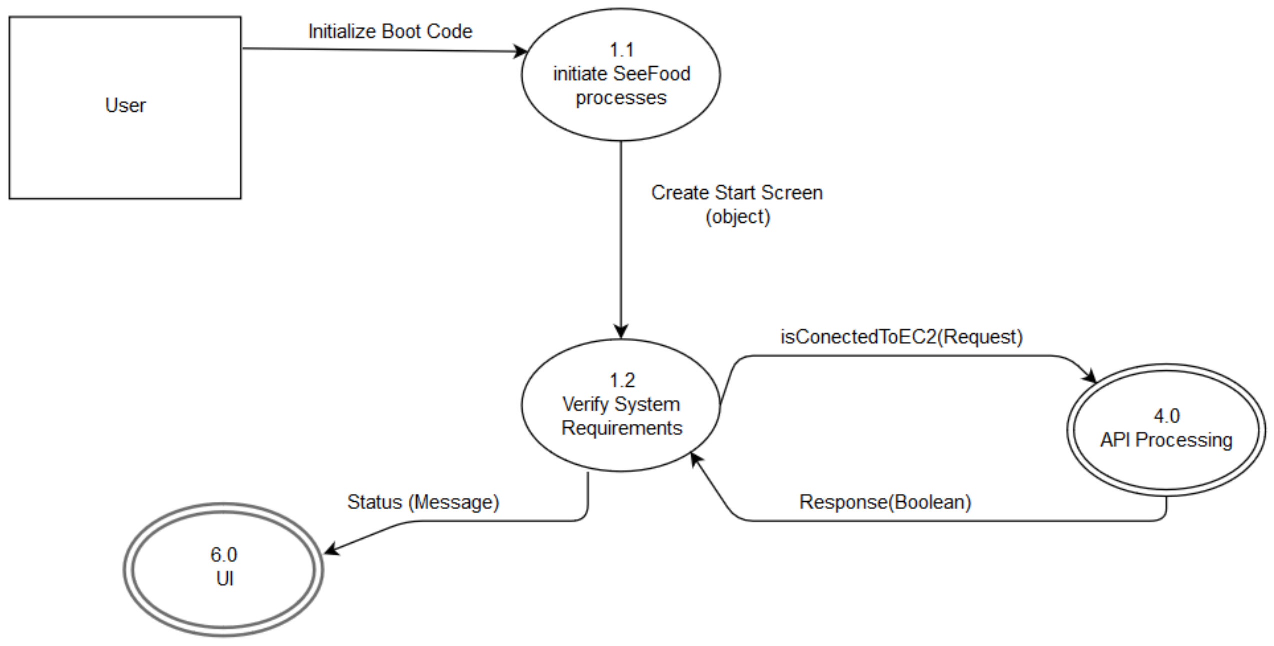


Mid-level DFD



Low-level DFD

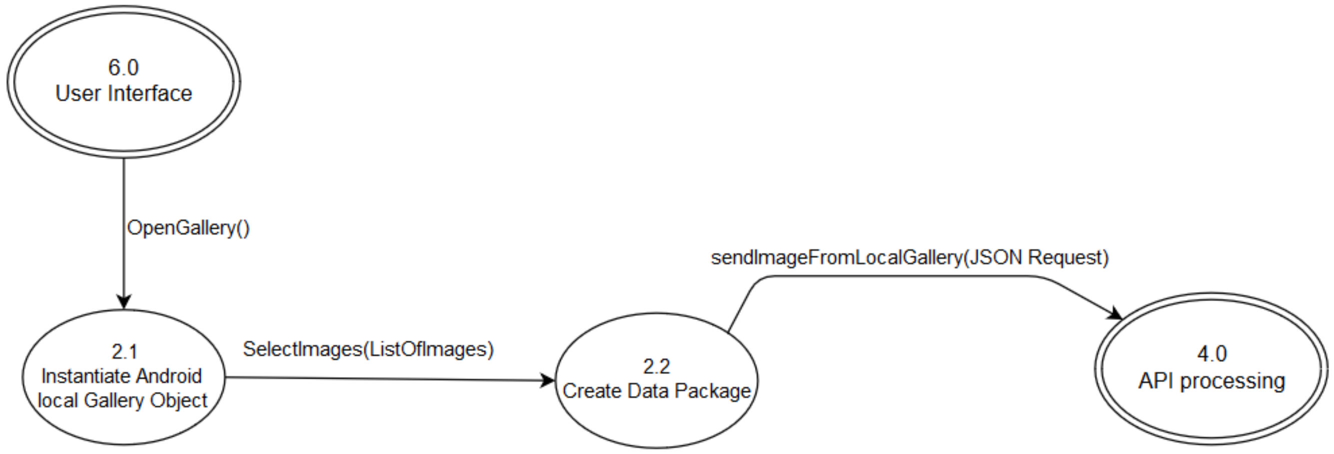
1.0 Startup Process DFD



Explanation

When the user opens the application, the initiation process (1.1) begins by displaying a start screen. The “verify system requirements” process (1.2) is called, and sends a message to the API Processing process (4.0) to ascertain whether or not the app can attain connectivity. If the server is online, a response message is sent and the process passes control to the UI process (6.0) with an “OK” message. If no response message is received in five seconds, the process will pass control to the UI process with a “Not OK” message.

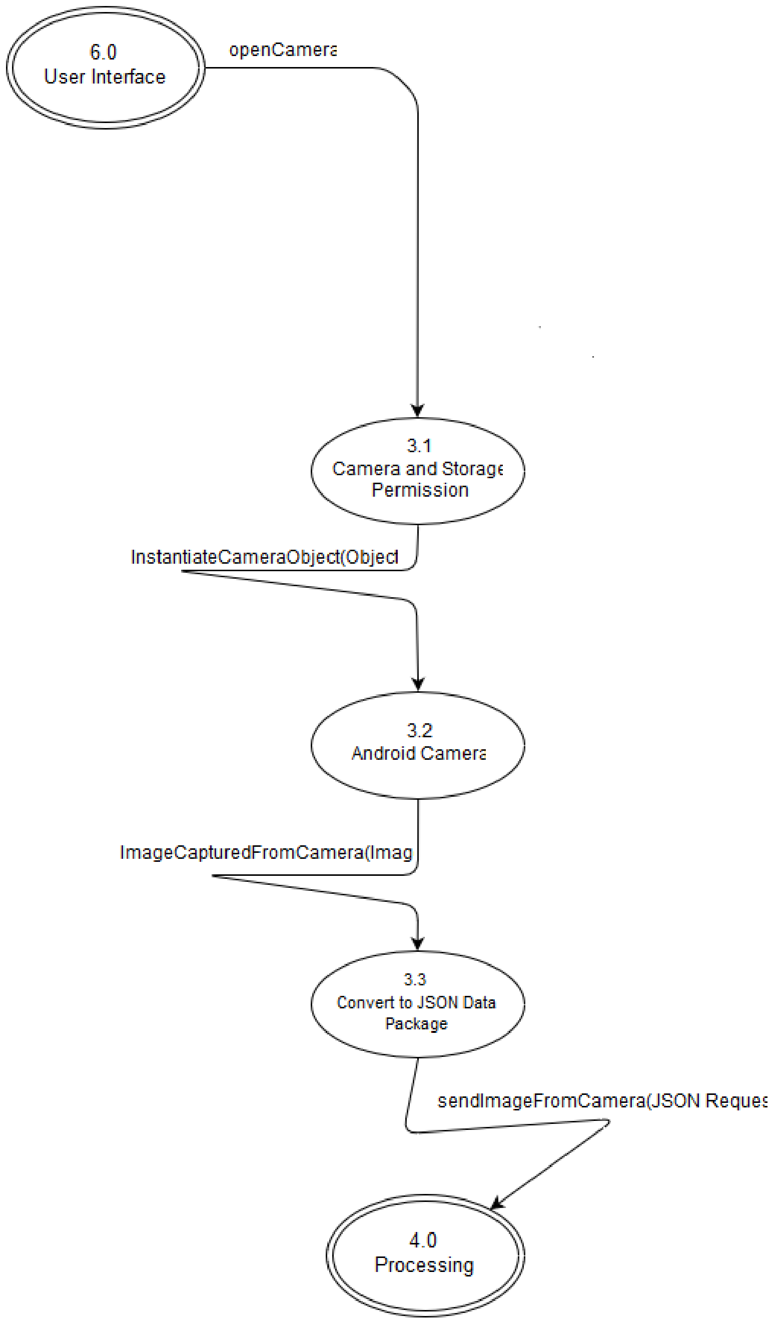
2.0 Local Gallery DFD


6.0
User Interface
OpenGallery()
2.1
Instantiate Android local Gallery Object
SelectImages(ListOfImages)
2.2
Create Data Package
sendImageFromLocalGallery(JSON Request)
4.0
API processing

Explanation

When the user opens the local gallery from the UI process (6.0), the app wills instantiate a local Android gallery object (process 2.1). The user may then select one or more images from this gallery for evaluation. When the user is done selecting images, he or she will indicate that the selection process is finished, and the image or list of images will be passed to the Create Data Package process (2.2). The image(s) will then be converted into a JSON data package and sent to the API processing process (4.0).

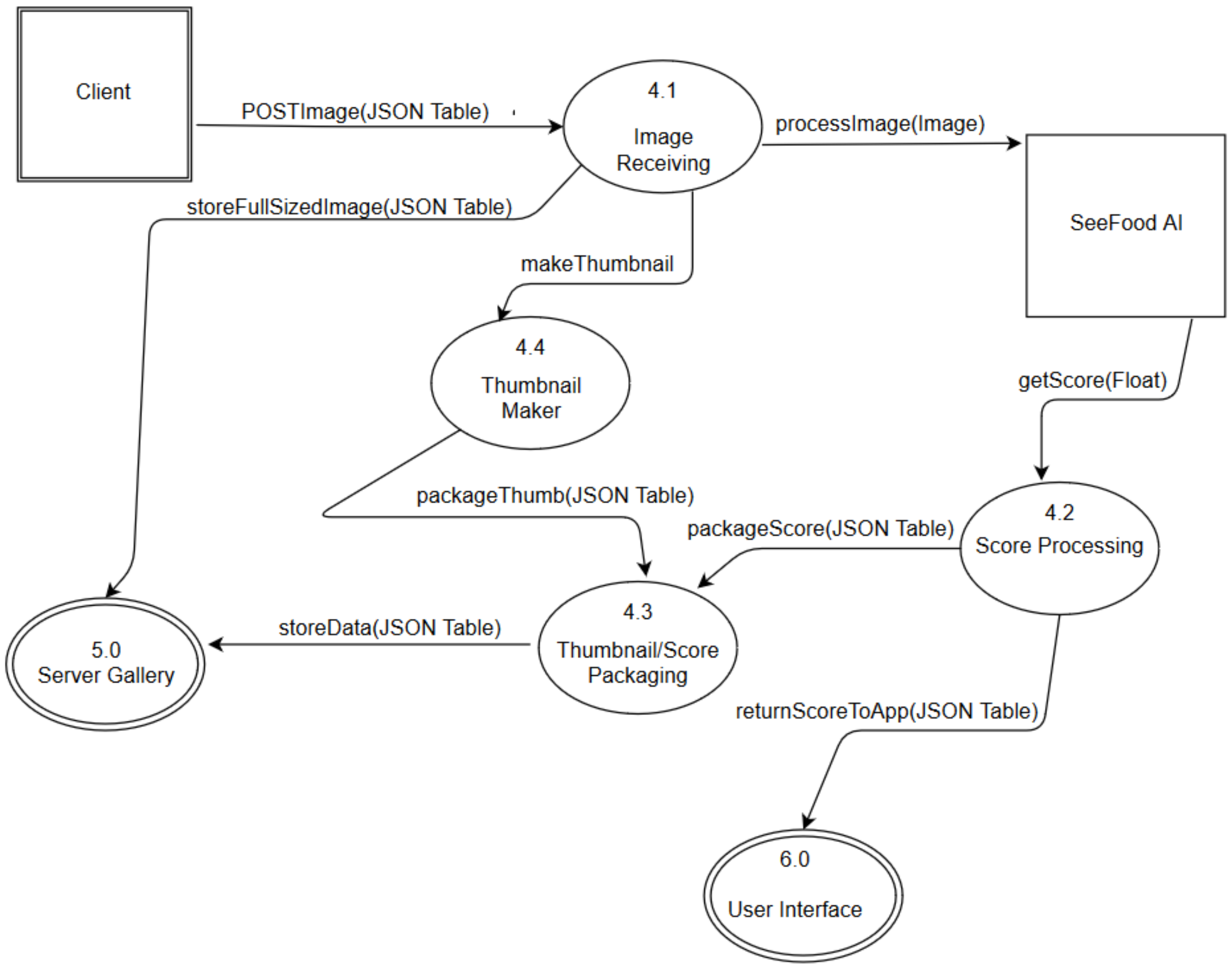
3.0 Camera



Explanation

When the user selects "Camera" from the UI process (6.0), the Handler Camera process is called (3.1). This process then calls the Camera and Storage Permission process (3.2), which establishes user permission to access the device's camera. The Android Camera process (3.3) begins, and the user takes a picture, which is sent to the Convert to JSON Data Package (3.4). This process sends the image, in JSON package format, to the API Processing process (4.0).

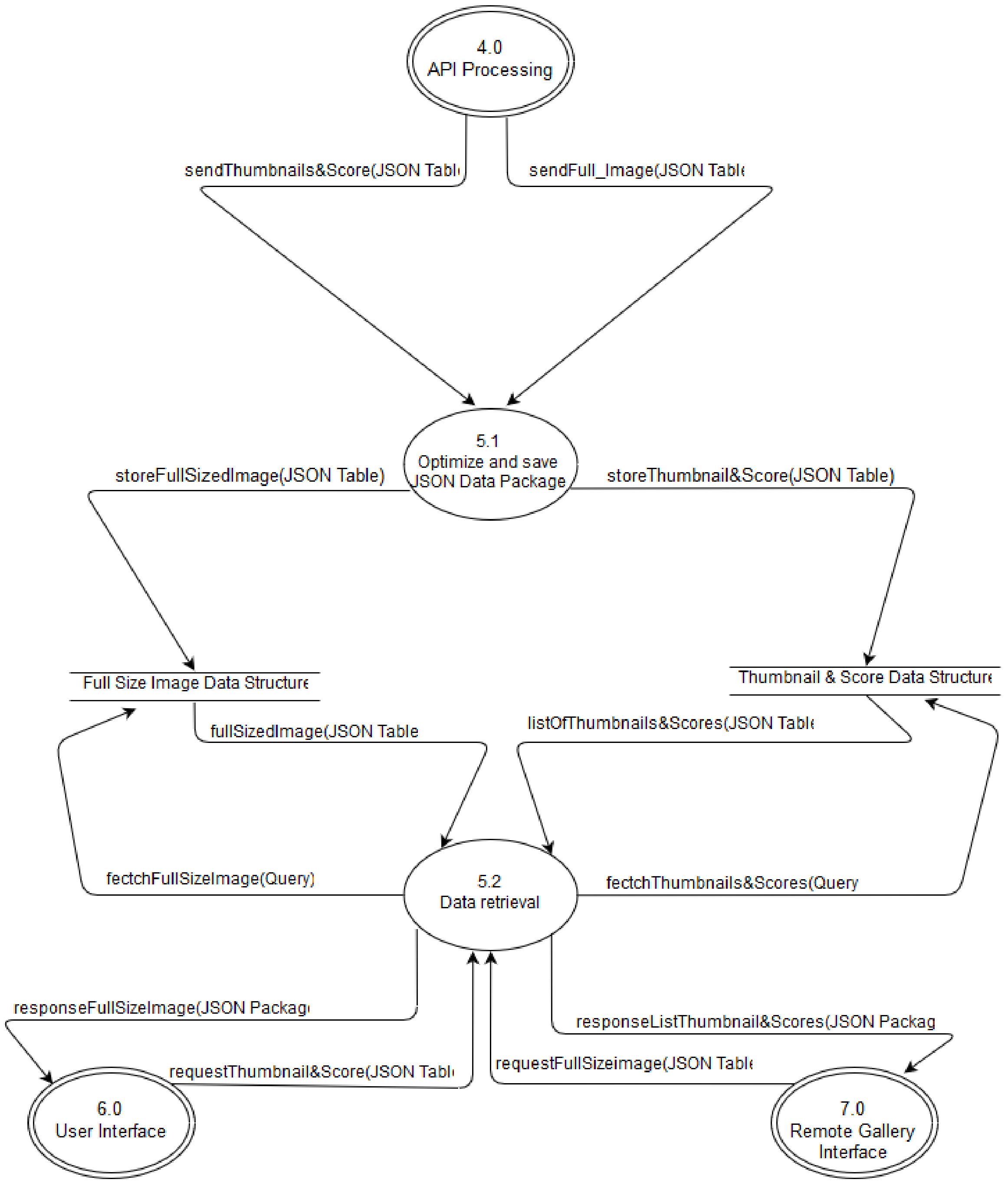
4.0 API Processing



Explanation

When the user sends an image to the server for processing, it is received by the Image Receiving process (4.1). This process forwards the image to three places: the Server Gallery (5.0) for storage, the Thumbnail Maker process (4.4), and the SeeFood AI itself for evaluation. The Thumbnail Maker converts the image to a thumbnail and sends it to the Thumbnail/Score Packaging process (4.3), where it waits for a score with which to pair. The SeeFood AI receives the image and returns a food score to the Score Processing (4.2). This process will convert the AI's score, which is a difficult-to-comprehend float between -2.0 and 2.0, into a more human-readable integer value between 0 and 100. This score will then be transmitted back to the User Interface process (6.0), as well as to the Thumbnail/Score Packaging process (4.3) which, you'll recall, has a copy of the thumbnail generated by the Thumbnail Maker process. At this point the Thumbnail/Score Packaging process has both a thumbnail and a score, and it packages these together in a JSON data package. This package is then sent to the Remote Gallery process.

5.0 Server Gallery

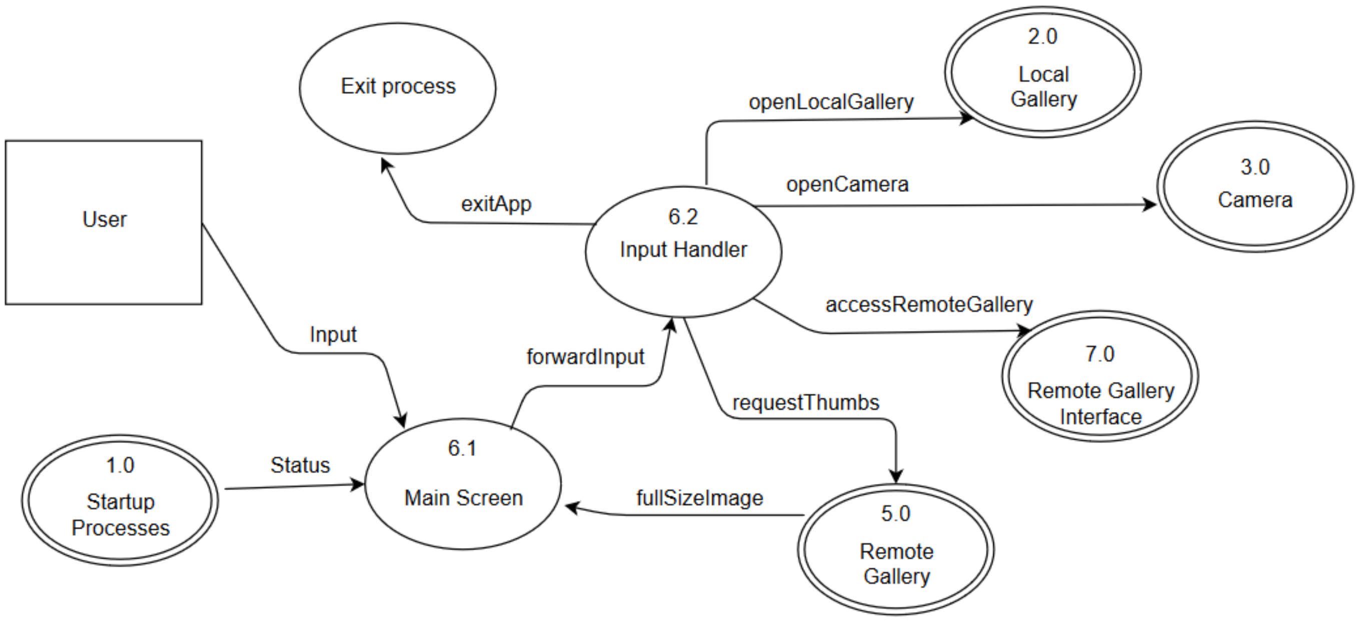

4.0
API Processing
sendThumbnails&Score(JSON Tabl
sendFull_Image(JSON Table
5.1
Optimize and save
JSON Data Package
storeFullSizedImage(JSON Table)
storeThumbnail&Score(JSON Table)
Full Size Image Data Structure
Thumbnail & Score Data Structure
fullSizedImage(JSON Table
listOfThumbnails&Scores(JSON Table
5.2
Data retrieval
fectchFullSizeImage(Query)
fectchThumbnails&Scores(Query
responseFullSizeImage(JSON Packag
requestThumbnail&Score(JSON Tabl
requestFullSizeimage(JSON Table
responseListThumbnail&Scores(JSON Packag
6.0
User Interface
7.0
Remote Gallery
Interface

Explanation

The API processing process (4.0), upon receiving an image or set of images in JSON package format from the client, forwards that files to the Save JSON Data process (5.1). Each image will be sent in both full-size format, and thumbnail format with an associated food score. Process 5.1 stores the full-sized image in a data structure, and the thumbnail/score package in a separate data structure.

The Data Retrieval process (5.2) receives the request for the complete set of thumbnail image/score objects from the User Interface process (6.0), retrieves all said objects from the thumbnail/score data structure, assembles a JSON package comprised of the objects, and sends it to the Remote Gallery Interface package (7.0). Similarly, when the Remote Gallery process sends a request for a full-sized image, Data Retrieval retrieves the image from the full-sized-image data structure, converts the requested image into the appropriate JSON format and sends it to the User Interface process.

6.0 User Interface

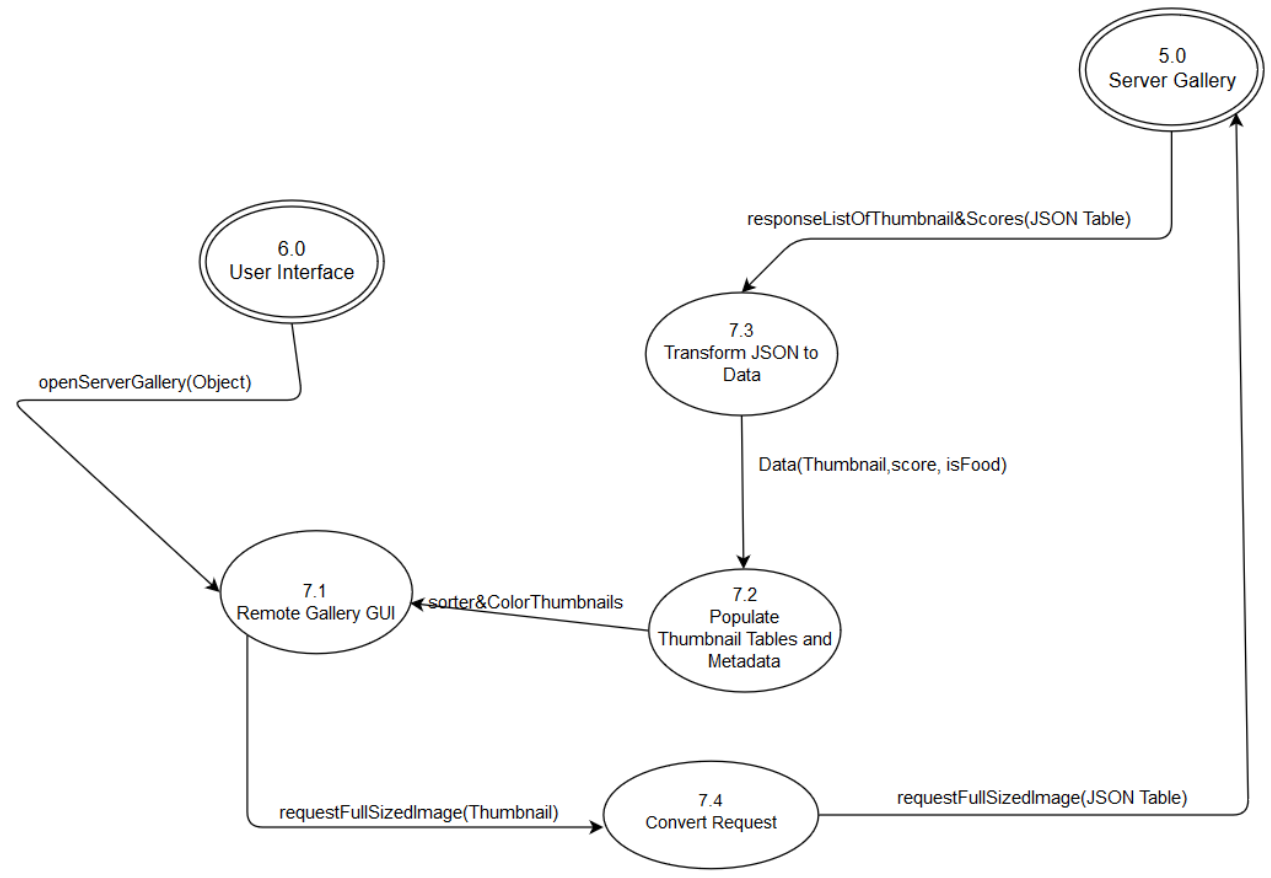


Explanation

Once the startup process receives a status message from the server, it passes control of the screen, along with the status message, to the Main Screen process (6.1). If the status message is “Not OK”, then the Main Screen process will display an error message and the app will have no functionality. If the status message is “OK”, the Main Screen process will open to the main menu, which will provide the user with options to 1. Open the phone’s local gallery (2.0); 2. Open the camera process (3.0); 3. Open the remote gallery interface process (7.0); and 4. Exit the app. The user’s touchscreen input will be forwarded to the Input Handler process (6.2), which will implement functions calling each of these processes respectively.

When the remote gallery interface process is opened, the input handler immediately sends a request to the remote gallery for the thumbnail/score objects stored within. This should minimize loading time for the thumbnail gallery. Furthermore, when the user sends an image to the server for scoring or uses the gallery to request a full-sized image from the server, the image is sent to the Main Screen process and displayed in the main area, displacing the previously displayed image (a hotdog logo by default). In the case that the user has uploaded multiple images, they will be displayed one at a time, and the user will navigate between them by swiping.

7.0 Remote Gallery Interface


5.0
Server Gallery
responseListOfThumbnail&Scores(JSON Table)
6.0
User Interface
7.3
Transform JSON to Data
openServerGallery(Object)
Data(Thumbnail,score, isFood)
7.1
Remote Gallery GUI
sorter&ColorThumbnails
7.2
Populate Thumbnail Tables and Metadata
7.4
Convert Request
requestFullSizedImage(Thumbnail)
requestFullSizedImage(JSON Table)

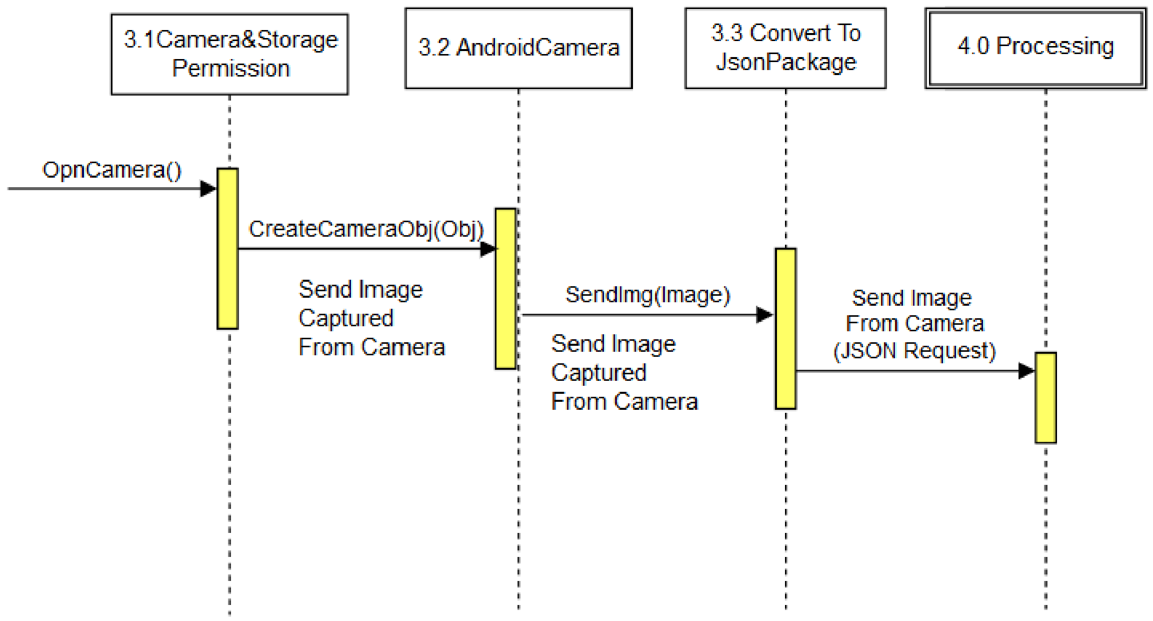
Explanation

When the user selects "Open Remote Gallery" from the User Interface package (6.0), the system passes control to the Remote Gallery UI process (7.1). Simultaneously, package 6.0 sends a request to the Remote Gallery package (5.0) for the complete population of thumbnail/score JSON objects, which is received by the Transform JSON to Data process (7.3). This process transforms the JSON files into image thumbnails and food scores, and passes this data to the Populate Thumbnails and Metadata process (7.2). This process inserts the thumbnails and food scores into a table, and colors the food scores along a spectrum from red to green according to the value of the score, with the lowest score (0) being red and the highest score (100) being green. The table is then passed to the Remote Gallery UI process.

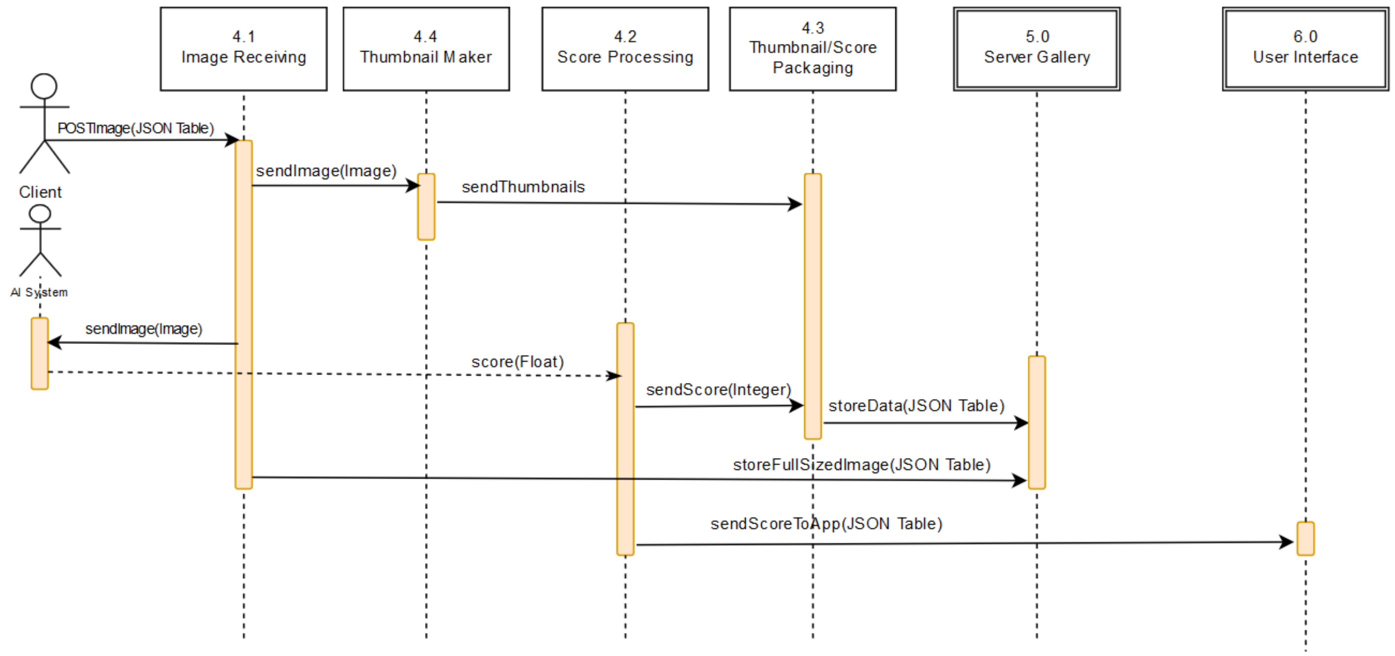
The user may then request one full-sized image by selecting the thumbnail from the gallery. The selected thumbnail is passed to the Convert Request process (7.4), which manufactures a JSON table request that is then sent to the Remote Gallery package.

Low-Level DFD Sequence Diagram

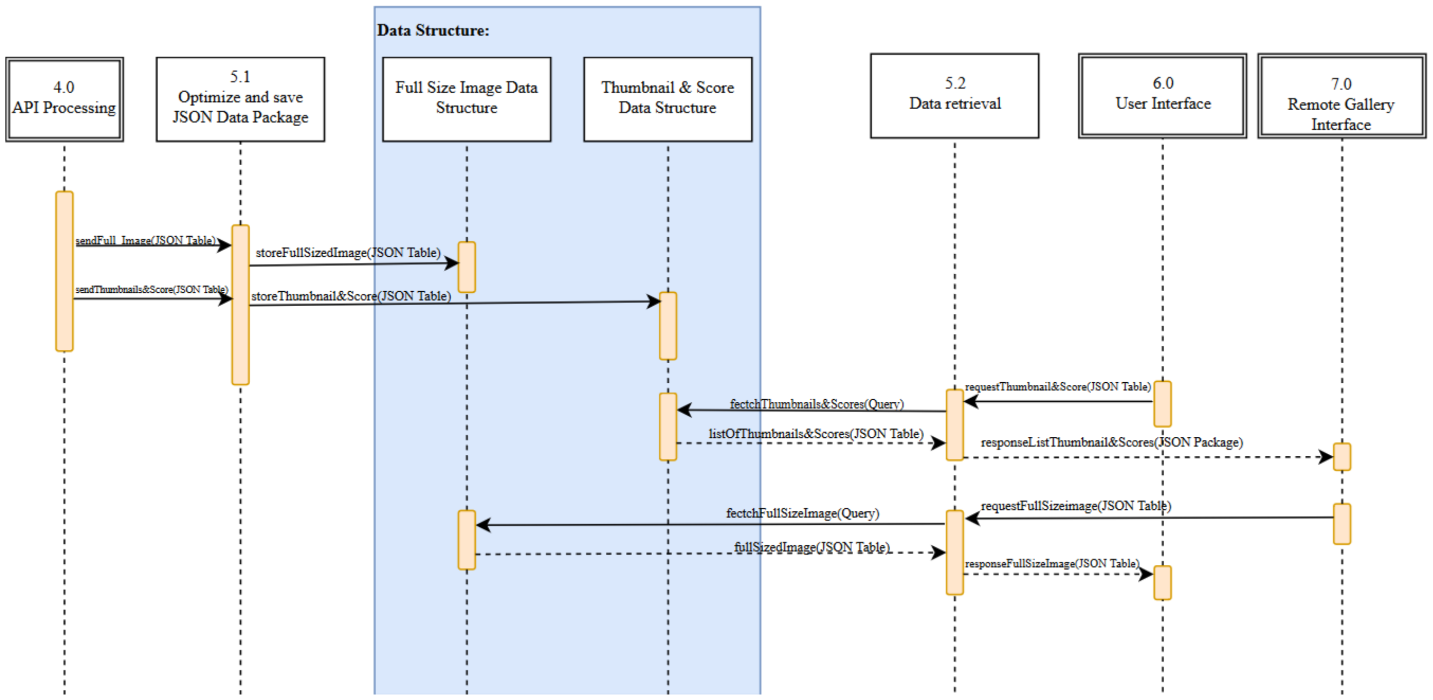
2.0 Camera Sequence


3.1Camera&Storage Permission
3.2 AndroidCamera
3.3 Convert To JsonPackage
4.0 Processing
OpnCamera()
CreateCameraObj(Obj)
Send Image Captured From Camera
SendImg(Image)
Send Image Captured From Camera
Send Image From Camera (JSON Request)

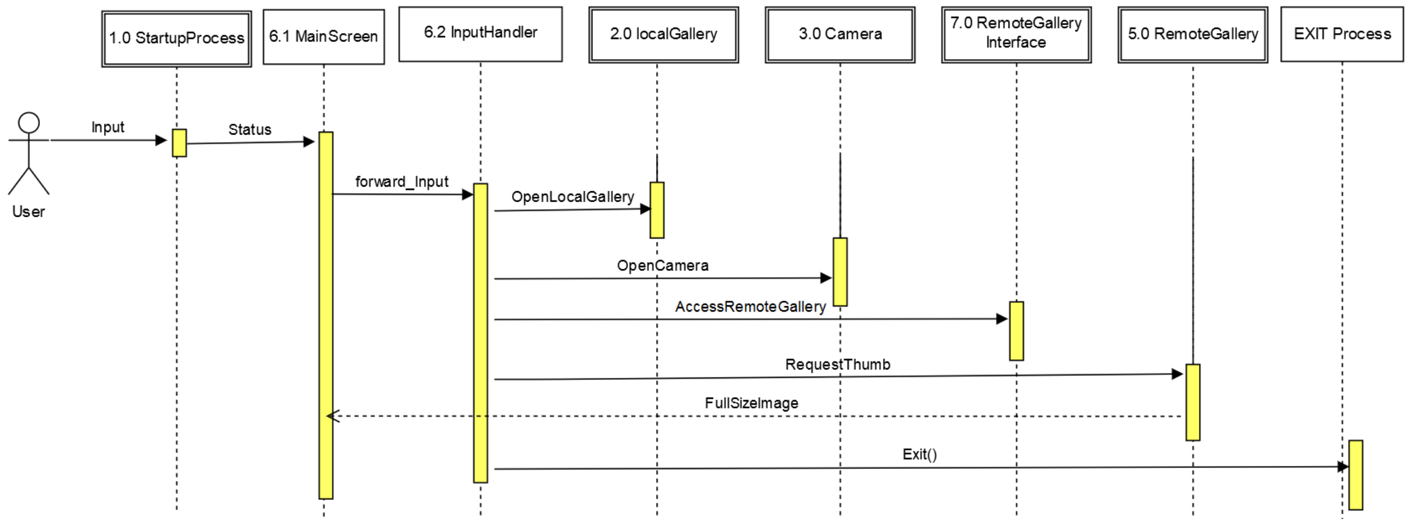
4.0 API Processing Sequence



5.0 Server Gallery Sequence

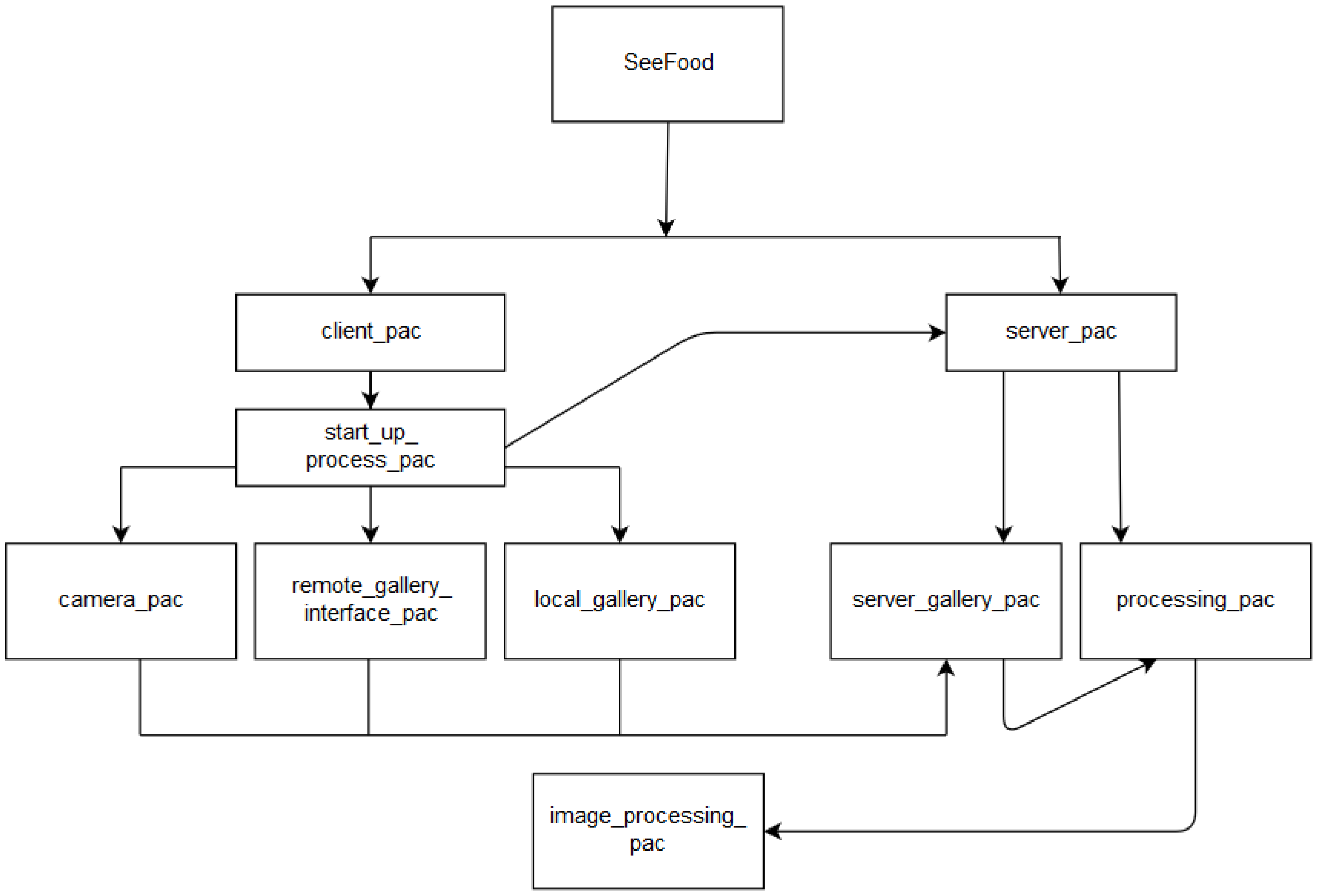

Data Structure:
4.0 API Processing
5.1 Optimize and save JSON Data Package
Full Size Image Data Structure
Thumbnail & Score Data Structure
5.2 Data retrieval
6.0 User Interface
7.0 Remote Gallery Interface
sendFull_Image(JSON Table)
storeFullSizedImage(JSON Table)
sendThumbnails&Score(JSON Table)
storeThumbnail&Score(JSON Table)
requestThumbnail&Score(JSON Table)
fectchThumbnails&Scores(Query)
listOfThumbnails&Scores(JSON Table)
responseListThumbnail&Scores(JSON Package)
requestFullSizeimage(JSON Table)
fectchFullSizeImage(Query)
fullSizedImage(JSON Table)
responseFullSizeImage(JSON Table)

6.0 User Interface Sequence



Software Design

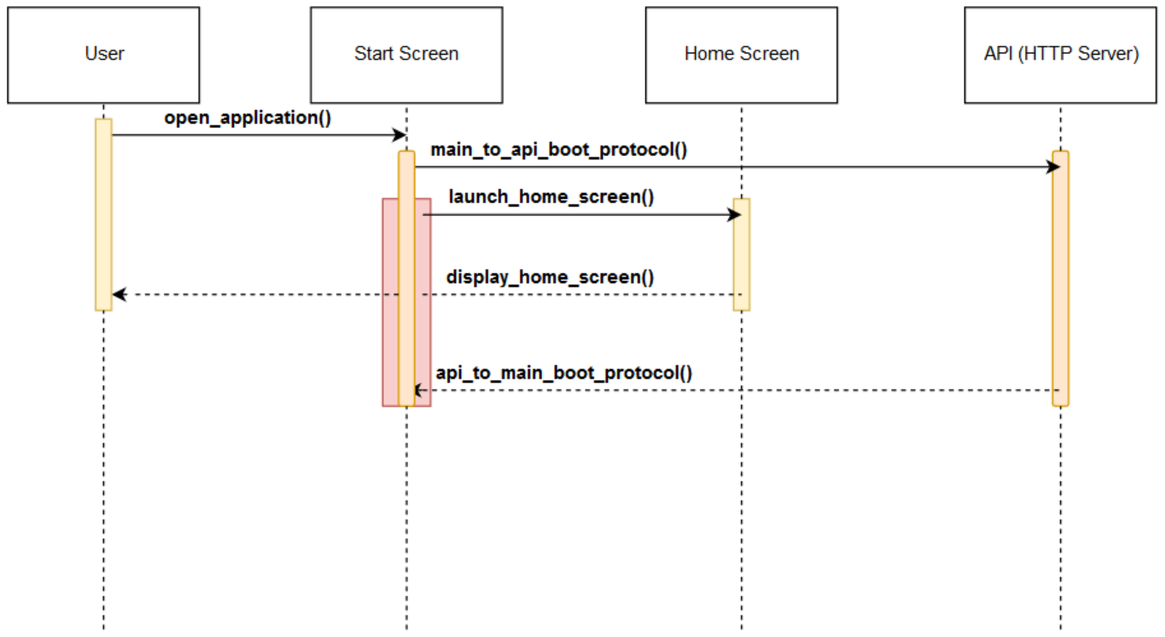
Package Diagram and Description



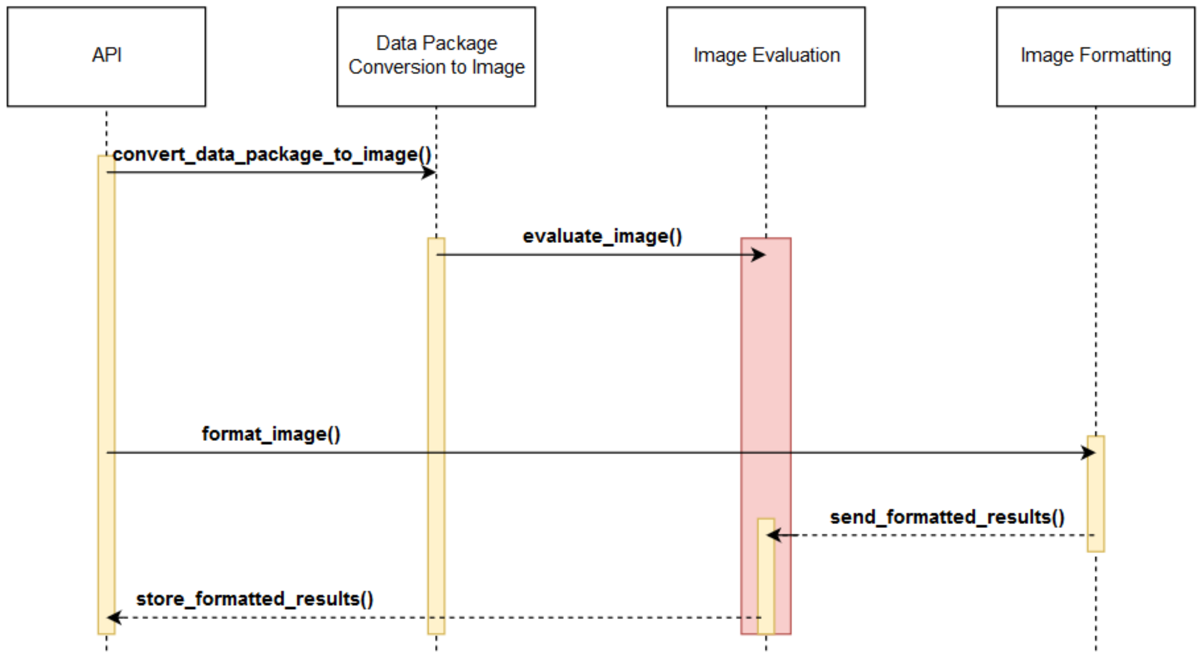
The SeeFood application has a client side application package and the server side package. On the client side, the start-up process boots the UI and the server side. Depending on the user's interaction with the UI, the client package will encompass a camera package for when the user wants to take a picture of an object/subject, a local gallery package that will store the processed image result, and a remote gallery interface package that gain access to the server side of the application for image processing. Now, on the server side, the server package will encompass a server gallery package that will store the images to be processed later on, and a processing package that will take in the images to be processed. Finally from the client side, the images being sent over to the server side will pass through the server gallery package, then be processed.

Use Case Sequence Diagrams

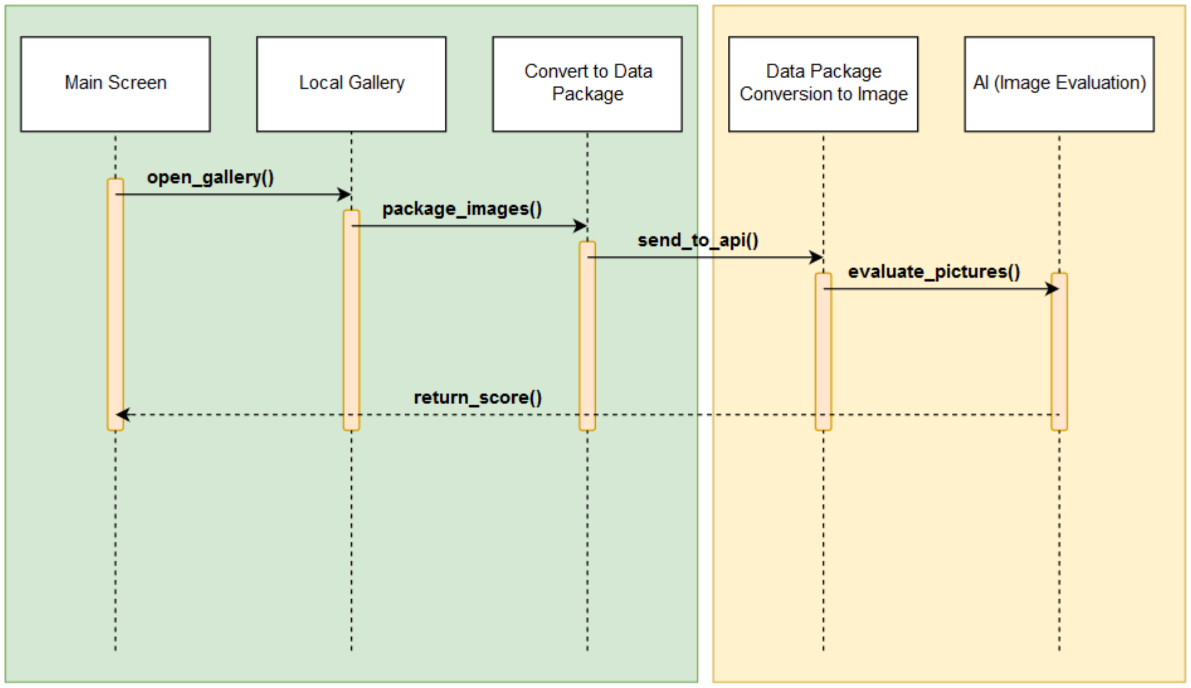
Sequence Diagram of Start Screen Use Case

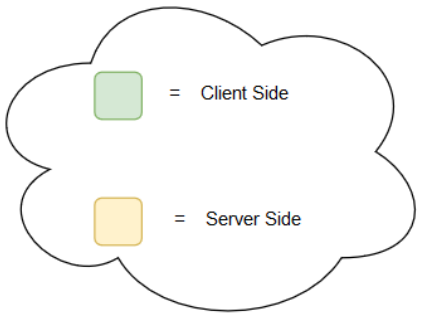


Sequence Diagram of EC2 Server Image Evaluation Use Case



Sequence Diagram of Local Gallery Use Case





Development Plan

Project Structure

We will divide our team into two groups. Don Miller and Ryan Zink will work together to develop the Android application and will be responsible for meeting all deadlines and specifications associated with the client software, and Nat Cross and Muhammad Khan will work together to develop the server software and will be responsible for meeting all deadlines and specifications for it.
Our team will primarily be communicating via a dedicated channel on Slack. We will also be holding informal meetings before and after class on Tuesdays and Thursdays, either in the fishbowl or the dedicated group projects space (348 Russ). We don't plan to use an additional project management tool, as we will be in frequent communication on GitHub, over Slack and in person.

Version Control

The main GitHub repository for our project will be located athttps://github.com/natcrossman/SeeFoodSystem, but each team member will fork this repository and be held equally responsible for maintaining version control.
Each team member will push to his local repo as he sees fit, but pull requests/major changes to the master branch will mainly occur upon issue resolution or major functionality additions.

Documentation Plan

The server team (team 1) will be producing developer documentation geared toward readers with a strong technical background. These documents will provide any relevant license information, as well as record the development process and reasoning behind choices made. The target reader of this documentation will be the individual tasked with maintaining or otherwise interacting with the finished software.

The client team (team 2) will be utilizing Javadoc functionality to document their code. These Javadoc comments will be aimed toward readers with a technical background, and include license information and clearly readable notes on the code and processes implemented in the software. Additionally, we will be producing a user manual in the form of a plain text file. This document will provide instructions on how to properly use the app to the end user. The instructions will be aimed toward readers without a technical background, and readability will be emphasized over precision/verbosity.

Project Schedule

Note: the team composed of Nat Crossman and Muhammad Khan is referred to as team 1, and the team composed of Don Miller and Ryan Zink is referred to as team 2.

Team	Milestone	Date	Notes
1, 2	Implementation begins	10-27-17	Coding begins.
1, 2	Send Single Image/ Receive Score functionality achieved	11-10-17	Core functionality is implemented on both client and server.
1	Server Gallery functional	11-12-17	Images are stored on and retrievable from the server.
2	Thumbnail gallery UI functional	11-15-17	The client can access and retrieve from the server gallery.
1,2	β Checkpoint	11-16-17	Project deliverable is turned in.
2	In-app camera functional	11-20	The camera should be the final functional addition to the app.
1	Server software finalized	11-20	The server-side processes are complete.
1,2	Testing	11-26	If any issues/bugs are discovered they are quickly resolved.
2	Build .exe file	11-28	The software is finalized and packaged.
1,2	Presentation/demo	11-28	Team 14 presents the app to the class.