InstaDam Documentation

Front end Team	**Back end Team**
Vedhus Hoskere	Zonglin Li
Fouad Amer	Wanxian Yang
Doug Friedel	Nishanth Salinamakki
Prahlad Srikant	Yu Tang

Table of Content

Contents

Table of Content ii
InstaDam 3
User Manual 3
User Management: 3
Projects: 4
Annotating Images: 5
Drawing: 6
Backend Deployment: 7
Development Process 8
Requirements and Specifications 9
Architecture and Design 10
WebAssembly Development 18
Reflections and Lessons Learned 18

InstaDam

Semantic segmentation is the process of classifying each pixel in an image into a given category or label thereby segmenting the entire image. Generating ground truth labels for training deep neural networks for semantic segmentation is a tedious task. InstaDam is a desktop application to create ground truth pixel-wise labels for deep learning based semantic segmentation of complex shapes like mechanical/structural defects in civil infrastructure like buildings and bridges or medical images in a user-friendly and semi-automated manner.

In currently available applications, typically, the boundaries of segments are created using a ‘lasso’ tool where the user clicks on one point at a time around the object of interest. This works well for rectilinear objects like buildings and roads. However, for complex amorphous shapes like civil engineering defects (Ex: cracks in pavements, bridges, or corrosion patches) or in medical images (Ex: MRI scans of brains or blood vessels in eyes) this requires too many clicks from the user all of which must be extremely precise.

Prior to the use of deep learning several image processing methods were developed to generate the segments. These traditional techniques are based on filters which have tunable thresholds to generate the segments. However, these techniques do not always give the best results by themselves when used in an automated fashion. InstaDam leverages the use of traditional image processing methods to help the user generate masks that can speed the process of generating accurate ground truth labels. The user will be able to select from several proposed image processing techniques and can visually examine the generated masks. Once the mask is selected then the user will use different input tools like paintbrush or the lasso tool to make sub-selections that correspond the required segments.

User Manual

User Management:

Registration

To register, you need to provide an email account, a password that has both lowercase and uppercase letters, longer than 8 characters, and has a special character. After providing the required information, you can press on **“Register”** to register a new account and login to it.

Login

(Server Version Only):

Logging in requires a deployed InstaDam backend server (please check the following section to learn how to deploy an InstaDam backend server) and a registered account. If these conditions are met, you can fill in the slots for your username and password along with the backend URL and press on **“Login”** to launch the main InstaDam window. If you already have a deployed backend, but you do not have an account, press on **“Register”** to create a new account.

Projects:

Creating a New Project

A project means defines the labels that will be used for annotations. These labels are defined by their names and colors. Define. To create a project, you can press on **“Project → New”** on the main menu bar. This will prompt a Label creation window. You can press on **“Add New Label”** to add as many labels to the project as needed. **“Add New Label”** will start a label creation dialog where you get to name the label and choose its color. After being done adding all the required labels, press on **“Ok”**.Local Version: After pressing on Ok, the project is not automatically saved, and you need to automatically press on **“Project → Save”** to save pick a saving a directory and providing the project name. Server Version: After pressing **“Ok”**, you will be prompted to type a name for your project. Press on **“Save”** to save the newly created project to the server.

Opening a Project:

to open a predefined project, you can go to **“Project → Open”**: Local Version: you will be prompted to pick a project from your local file system. Server Version: a list of all the projects that you have access to will be displayed along with your privileges in each one of them. You can double click on any project to open it.

Saving a Project *(Local Version Only):*

To manually save a project, go to **“Project → Save”**. You will be prompted to provide a project name and pick the project saving destination.

Deleting a Project *(Server Version Only)***:**

This motion is irreversible in InstaDam and it will delete all the project labels, images, and annotations from the backend. To perform the action, go to **“Project → Delete”**. a list of all the projects that you have access to will be displayed. You can double click on any project where your privilege is Admin to delete that project. After double clicking, an alert message will ask you to confirm the deletion action. Press on **“yes, delete”** to finalize the deletion process. In case you deleted the current project, you will need to open another project afterwards to be able to use the software.

Adding User to a Project (Server Version Only):

In order to add a user to the current project, you can go to **"User → Search"**. You can type the user's username or email (or a portion of them) to search for a user. A list of matching users will be returned and displayed in the list. Pick on of the users and press on **"Add to Project"**. A dialogue box will ask you if you want to add the new user as admin or as annotator. Pick the required privilege and the user will be added accordingly.

Updating User Privilege in Current Project *(Server Version Only)*:

In order to update a user privilege in the current project, you can go to **"User → Search"**. You can type the user's username or email (or a portion of them) to search for a user. A list of matching users will be returned and displayed in the list. Pick on of the users and press on **"Update Privilege"**. A dialogue box will be displayed for you to pick the new privilege.

Adding Images to a Project *(Server Version)*:

Once a project has been opened, we can go to **"Annotations->Open"**. This will bring up a list of images associated with the project. You can upload a new image by click upload in the list dialog. If at least one image file is selected, it will be uploaded to the server and the list will updated to reflect this.

Opening Images *(Server Version)*:

Once a project has been opened, we can go to **"Annotations->Open Image**". This will bring up a list of images associated with the project. To open one, select the image, click the load button, and the canvas will be populated with the selected image and any annotations associated with the image.

Annotating Images:

Selection tools are used to annotate regions of images. There are four basic types of selection tools:

- Rectangle for selecting rectangular regions.
- Ellipse for selecting elliptical regions. Both rectangles and ellipses are described by an enclosing rectangular box and are interacted with in the same way.
- Polygon for irregularly shaped regions defined by vertices connected with straight lines.
- Free drawing is for defining regions with a brush. Both round and square brushes are available and the brush size can be adjusted.

Every selection region must be associated with a label so that it is properly annotated. Annotations can be drawn on either the photo (left) or mask (right) panes. The following sections describe how to use each of the tools.

Drawing:

Rectangle and Ellipses

In order to draw either of these shapes, select a label and the appropriate tool button (Rectangle or Ellipse). Left click to define the top left corner, then drag the mouse (while keeping the button depressed) to define the shape, and release the button to complete the selection.

Polygons

To draw a polygon region, select a Label and the Polygon button. Vertices of the polygon are denoted by clicking the left mouse button and can be added one after another by subsequent clicks. A vertex can also be added between two existing vertices by first clicking on the "Insert Point Between" button, then clicking on the two vertices which you want to insert a vertex between. Then click a point to add a vertex. Click the "Finish Polygon" button when done.

Free Draw

To use this tool, select a Label and the FreeDraw button. Chose the brush shape and size from the interface. Then left click and move the mouse (while holding the button down) to paint the region.

Moving

Rectangles, ellipses, and polygon objects can be moved around the canvas. To do this, hold down the “Shift” key and left click on an item. Hold both buttons down and drag the mouse to move the item to its new position. Then release the mouse button and key.

Resizing

Rectangles, ellipses, and polygon objects can be resized. To do this, hold down the “Shift” key and click to left mouse button on the object you want to resize. This will make the object active and put boxes at the vertices. While still holding the “Shift” key click and hold the left mouse button while the pointer is over a vertex. Drag the mouse to move the vertex to its new position and release the mouse button and key.

Rotating

Only Rectangle and Ellipse objects can be rotated. They are rotated about their center. This is done by holding the “shift” key and right clicking on the object and dragging the mouse (while keeping the button depressed) until the object reaches the desired rotation. Release the button to place the item.

Erasing

Rectangles, Ellipses, and Polygons can be removed from the canvas by holding the “Shift” key, left clicking to make the item active. The press the backspace key to delete the item. Individual vertices of a

polygon can be deleted by holding the “Shift” key down and left clicking on a vertex. The vertex box should be highlighted. Then press the backspace key to remove the vertex.
Free draw objects are not directly deleted, but painted with the Erase tool to remove portions of them. Click on the Erase button on the panel, select the desired brush type and size, then draw on the canvas with the left mouse button depressed. This will erase any points from free draw items (from the current label) that are under the brush.

Drawing on the Mask

When you draw annotations on the Mask pane (right hand side), only pixels which are not masked (white on the Mask) will be included in the final selection. For Rectangle, Ellipse, and Polygon items, draw on the Mask as normal, but you must click on the "Add Selection" button to compute the actual pixels selected and add them to the display. Once this button is clicked the original SelectItem is discarded for the new pixel selection. Clicking on the "Cancel Selection" button to discard the selected region before applying it.
For the Free draw the drawing is done as it is on the photo panel, however there is no need to add the selection as the masked pixels are calculated when the mouse button is released.

Undo and Redo

Any selection action (add, move, rotate, resize, erase) can be undone and redone by using the menu or the shortcut keys (Ctrl-z and Ctrl-Shift-Z).

In the case of drawing on the Mask, actions can be undone and redone for Rectangle, Ellipse, and Polygon objects before the "Add Selection" button is clicked. Once the selection has been merged with the mask the undo/redo actions will only remove or add back the computed pixels.

Backend Deployment:

1. Deploy with Docker:
    a. Set necessary environment variables for deployment:
        i. `DB_USERNAME`: Database username.
        ii. `DB_PASSWORD`: Database password.
        iii. `DB_NAME`: Database name for InstaDam app.
        iv. `SECRETE_KEY`: User supplied secrete key for the app.
        v. The default admin username and password is 'admin/AdminPassword0', you can change it in `instadam/config.py` before deployment.
    b. Run `docker-compose up` in project root folder
2. Deploy in custom environment:

a. First, you should have a PostgreSQL instance up and running on your server.
b. Then you need to set the corresponding environment variables for the InstaDam app:
    i. `_DB_USERNAME`: Database username.
    ii. `_DB_PASSWORD`: Database password.
    iii. `_DB_HOSTNAME`: Databse hostname
    iv. `_DB_NAME`: Database name for InstaDam app.
    v. `_SECRETE_KEY`: User supplied secrete key for the app.
    vi. The default admin username and password is 'admin/AdminPassword0', you can change it in `instadam/config.py` before deployment.
c. Deploy a production server:
    i. `python3 manage.py deploy`
    ii. The default behavior is to delete all previous data/table structure in the given database and reinitialize from ground up.
    iii. Alternatively, you can reuse the previous data by using `python3 manage.py start --mode=production`

Development Process

We adopt the Agile principle for our development process. We divided the work into iterations, each of which being as short as two weeks. To follow the Agile principle, we have set up continuous testing and continuous integration (i.e. TravisCI, Azure VMs) for the backend. Tests are run after every commit and before merging, to keep our master branch clean and functional. Similarly, frontend also set up automatic testing. To ensure code quality, we configured a set of code formatters, and are doing refactoring at the end of each iteration.

Since the project involves both frontend and backend work, our team is split accordingly. After the first iteration, we organized into smaller groups and started setting up the environment and servers. Collaborative development is done through Github via the Git VCS. Github allows comments and suggestions on pull requests, which is how team members recommend ways to refactor and structure code. There is occasionally miscommunication about specific files to edit, which could sometimes result in merge conflicts and thus leads to additional time to implement working user stories. However, because the group consists of a total of eight members, it is hard to get everyone together. So there are only one or two all-hands meetings every week. To make up for this, we rely more on group messaging apps to keep everyone up to date.

Requirements and Specifications

UC	Actor	Goal	Priority
1	Annotator	Be able to undo an annotation	2
2	Annotator	Be able to redo an undone annotation	2
3	Annotator	Open an image	1
4	Annotator	Load an image from the cloud into the program	1
5	Annotator	Display the canvas and annotation tools	1
6	Annotator	Browse and select an image to work with	1
7	Annotator	Create and Modify (erase) Mask	1
8	Annotator	Erase an existing Mask	1
9	Annotator	Use a certain annotation tool to create a Mask by selecting pixels on the screen	1
10	Annotator	Assign a label to a created Mask	1
11	Annotator	Erase an annotation	3
12	Annotator	Select a label for annotation	2
13	Annotator	Choosing which tool to use, and configuring the parameters of that tool	2
14	Annotator	Change the brush size being used for region selection	3
15	Annotator	Select a region with the lasso (polygon) selection tool	2
16	Annotator	Select a region with the box selection tool	2
17	Annotator	Select a region with the ellipse selection tool	2
18	Annotator	Choosing one of the available filters and configuring the parameters of that filter	2
19	Annotator	Choose a filter to use	1
20	Annotator	Set the filter options and apply them to the image	2
21	Annotator	Choose a label for the annotation	1
22	Annotator	Save the annotations to the project	2
23	Admin	Upload image to the server	1
24	Admin	Select an image to add to the project	1
25	Admin	Select images to export from the project	3

26	Admin	Choose export mask type for the dataset	3
27	Admin	Choose the file type for the project	2
28	Admin	Create a project	1
29	Admin	Define and add an new label	1
30	Admin	Create label names for the project	1
31	Admin	Choose label colors for the project	1
32	Admin	Choose filters to be available in the project	2
33	Admin	Providing access to other users (mainly annotators)	2
34	Admin	Generate user id's	2
35	Admin	Specify access control for each user	2
36	Annotator	Change opacity in viewer	2
37	Admin	Add user to the project	2

Architecture and Design

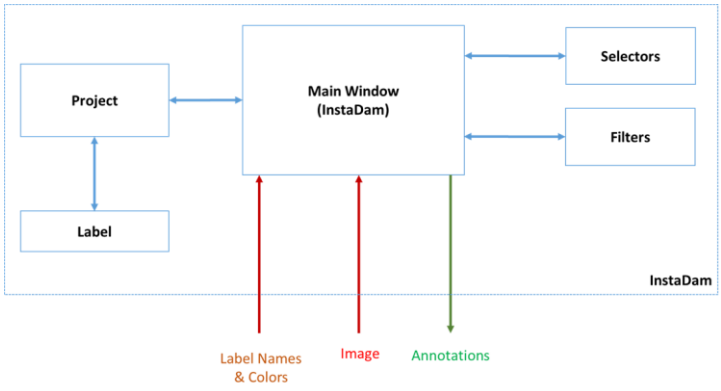
Our system is built using a three-tier architecture composed of a client layer, business layer, and data layer. We chose this architecture for several reasons:

1. Three-tier architecture provides high modularity between each layer. The client and backend code can be developed separately using its own frameworks and languages. This also makes the architecture flexible and easily extendable in the future.
2. This architecture is also easily scalable. For example, if the application is querying the database extremely often, we can simply scale the data layer by scaling up the Azure database cluster. We can also scale the application layer by adding more Azure web servers.
3. It also makes maintenance much easier. Refactoring the backend layer will not affect the client layer adversely and vice-versa.

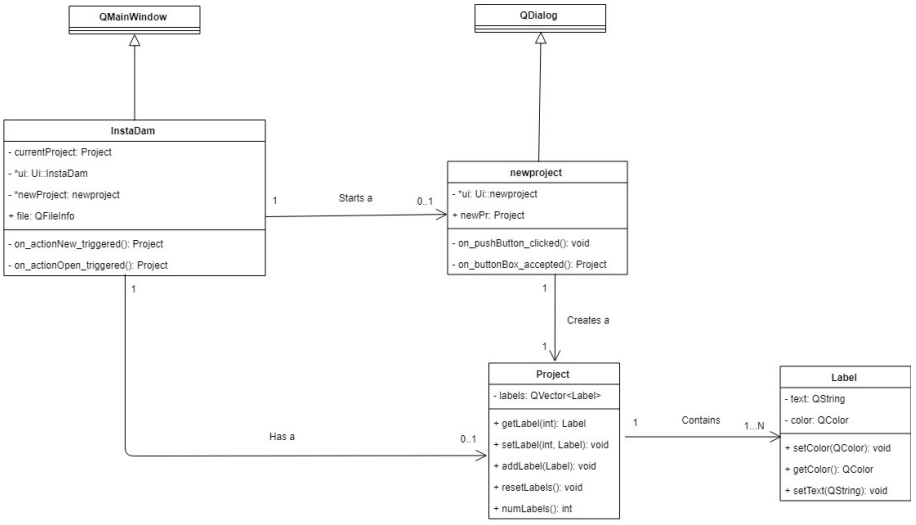
The client layer provides all the front-facing tools for the user to interact with the application. This includes creating new projects, opening existing projects and images, adding annotations with labels to the image, applying filters to the image, etc. Essentially, data (i.e. image bits, project details) are created or modified by the user via the client, and the client saves this information by making REST requests to the backend.

The client layer was built using C++ and QT. We chose QT for this project because we wanted a platform that could be used both as a desktop tool and as a web application, and Qt offers this flexibility by supporting WebAssembly. Our approach was therefore to build a desktop application that could be used as a standalone tool using local storage or as a remotely hosted application, and then to use Qt for WebAssembly to transform that desktop application into a web application.

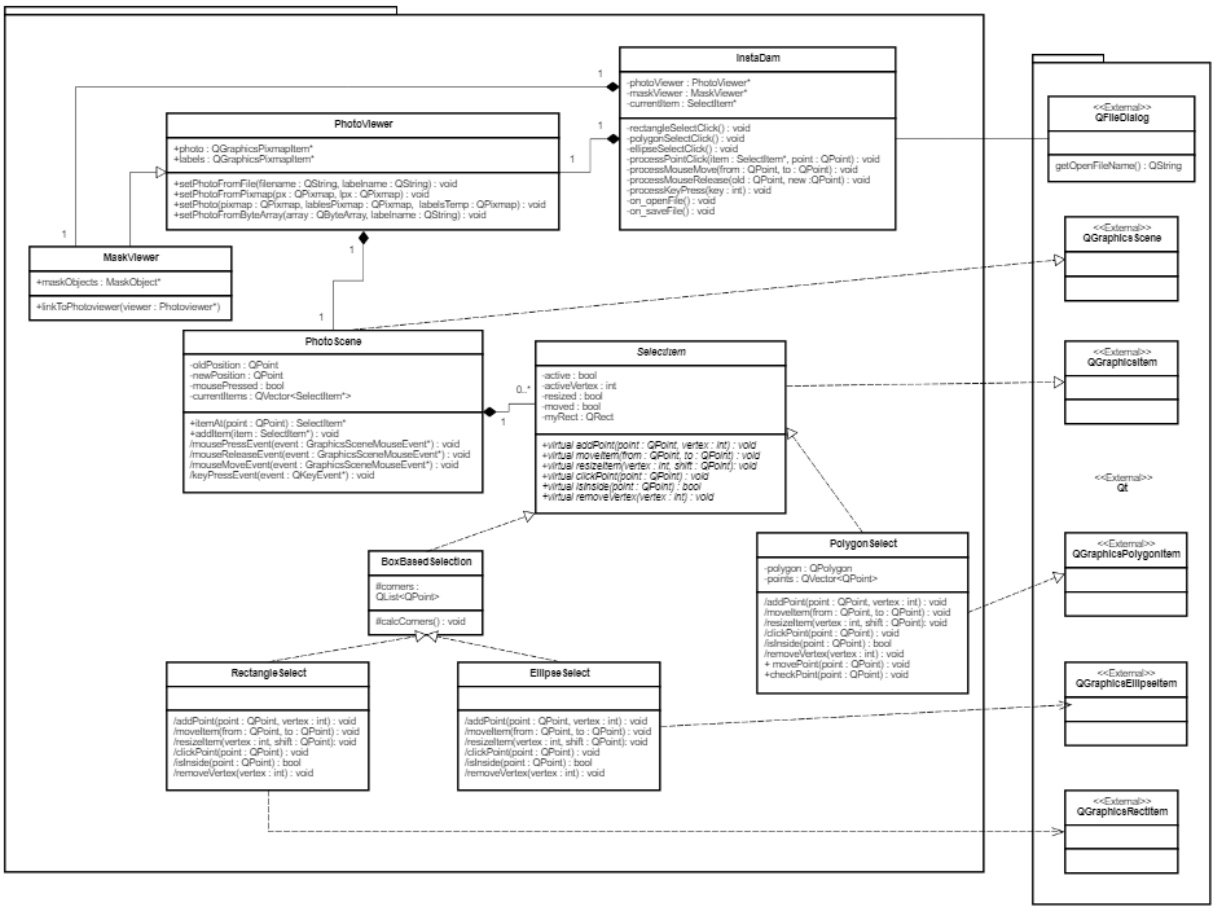
Conceptually, we define a "**project**" as a set of labels where each label has a name and a color and an "**annotation**" as a the markings (as pixel coordinates) made by the user to identify the location of a label segment. Additionally, when running on a server, each project is associated with a set of drawings and a list of users. We also define a "**filter**" as an image pre-processing function that creates an image "**mask**". In total, 3 different filter types are provided: "Blur", "Canny", and "Threshold". The created "**mask**" is the result of applying a filter to an input image. This masking step is used to fine-tune the boundaries of the drawn annotations by adding only the non-masked parts of a certain selection to the saved annotations. Moreover, we define a "**selector**" as a tool to create annotations. 4 different types of selectors are provided: "Box", "Polygon", "Ellipse", and "Free Select". In order to handle these different concepts, the architecture of the front end can be described by the following diagram:



Instadam Frontend Architecture



UML Diagram for Creating a Project on the Frontend



Class Diagram for Selection Tools

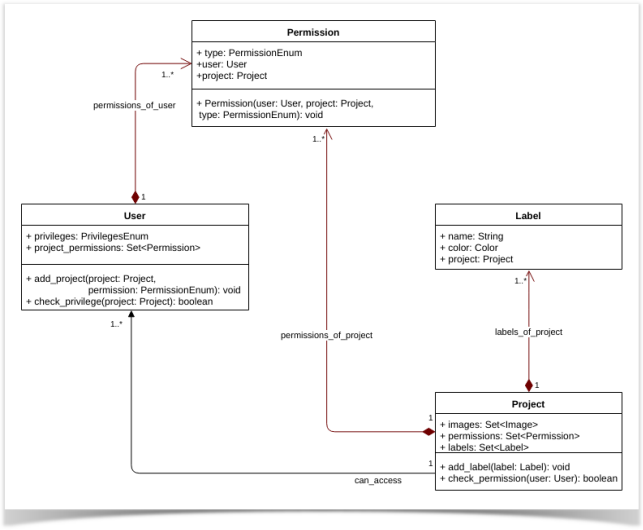
The backend layer accepts requests made by the client and processes it. This includes operations like storing data in the database layer, making queries to the database layer, and verifying tokens. It is responsible primarily for saving and persisting all the data the user creates or modifies on the client side

so that the projects and images can be accessed remotely from anywhere. There are also auxiliary functions it performs like user authentication and user-project permission verification.

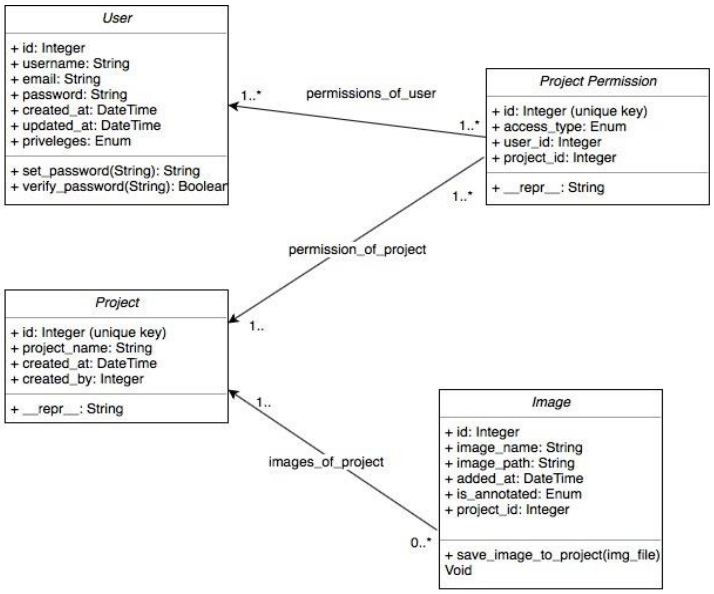
The backend layer was built using Python since Python is an extremely fast language to develop in with great support and a lot of online documentation. We used Flask as the web framework because it has an easier learning curve for people new to Python and is extremely flexible in terms of adding and modifying endpoints. In addition, we also used SQLAlchemy as our object relational model to store and query data from the database. We used an object relational model over a relational database model because it abstracts away the database with an object-oriented code interface. It also simplifies interaction with the database by removing the need for direct SQL statements, allowing for more secure code and easy cross-compatibility between different multiple SQL servers.

Using these frameworks influenced our design in many ways. The backend system is comprised of a few major components.
The first one contains all the models for all the objects in the database, such as the annotation, image, and project. The models are located in the /instadam/models directory. They use SQLAlchemy to define the structure for these new database tables for each object, but allow us to interact with the database (i.e. make queries, store data) via object-oriented code. Each model also has additional methods to abstract away common queries made for many user stories.

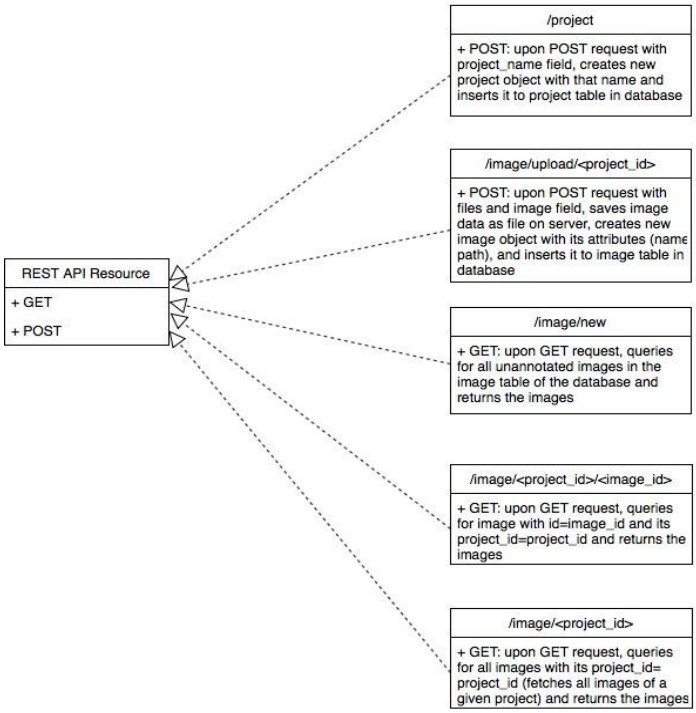
The second major component contains all the files that implement the user stories (i.e. saving project data, uploading images). These files are located in the /instadam directory and are independent from each other to keep the design as modular and self-contained as possible. All these files have Flask defined endpoints that interact with the client via REST API requests. Depending on the endpoint and the REST request type, the backend implements the corresponding user story. All these processes are essentially a list of sequential operations that involve multiple interactions between the application and database layer. Below are UML diagrams depicting most of the implemented user stories:



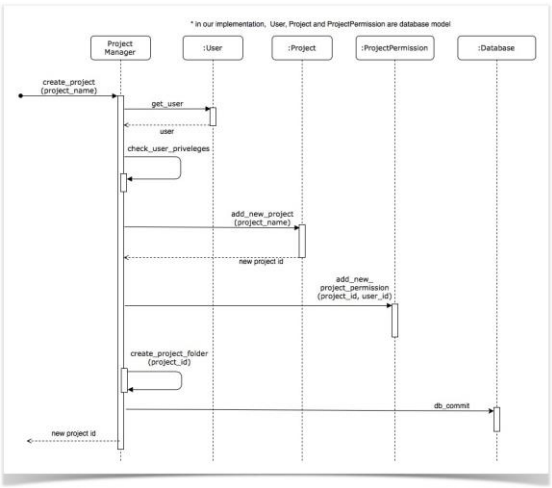
UML Class Diagrams -- Add Label, Specify Access Control



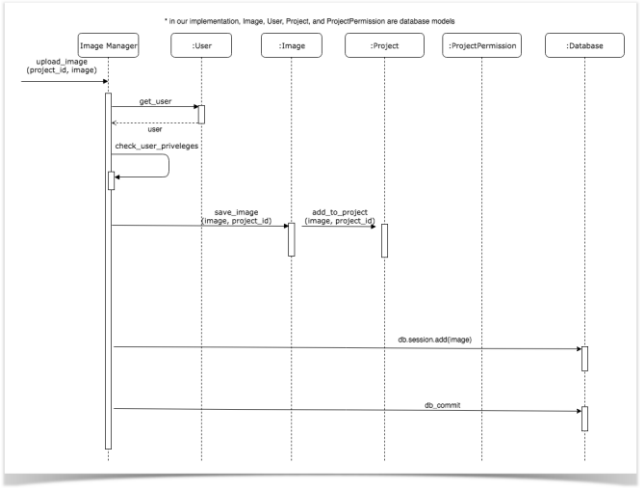
In addition, our use cases are implemented through the use of REST API endpoints. Below is another diagram to illustrate this part (as stated by Professor Marinov when asked during class). Although it might not be considered as a class diagram since technically the endpoints are not classes, but the diagram is a good illustration of how the endpoints are designed and how they interact with the classes that are drawn above.



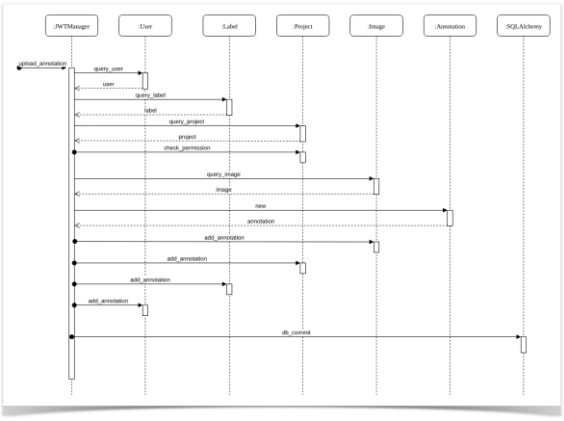
UML Class Diagrams -- Load Image, Upload Image, Create Project



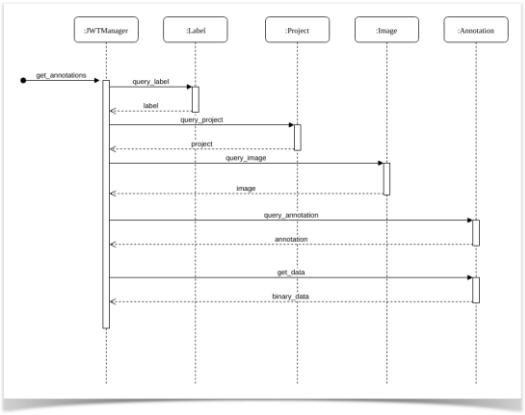
UML Sequence Diagrams -- Create Project



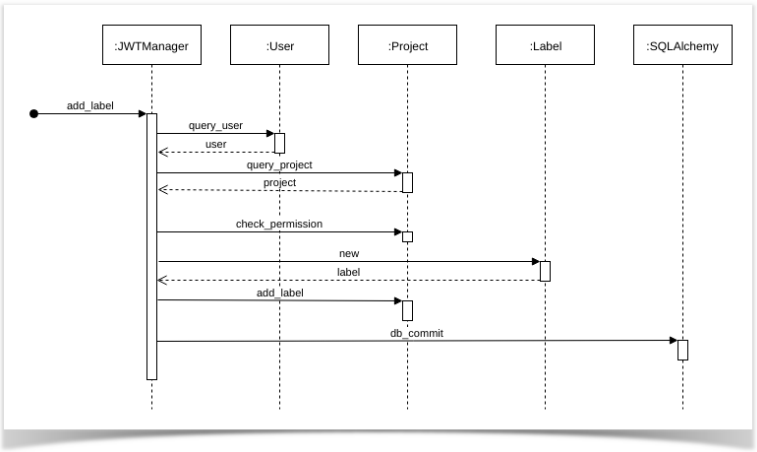
UML Sequence Diagram -- Upload Image



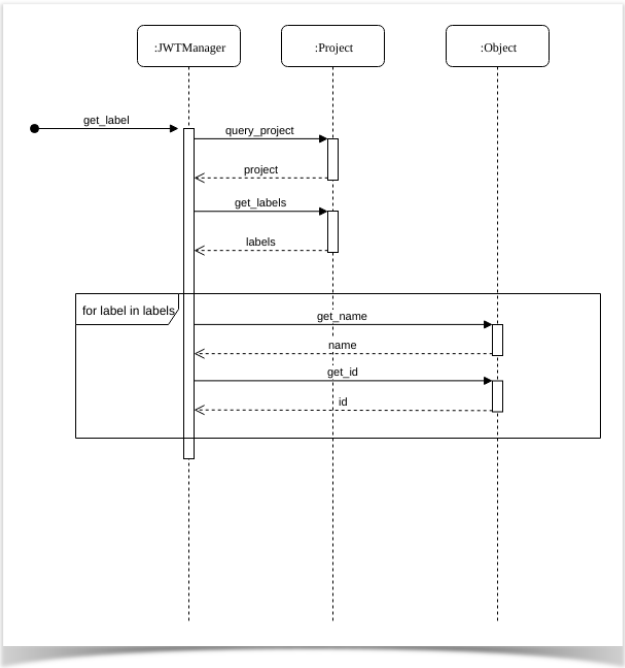
UML Sequence Diagram -- Add Annotation



UML Sequence Diagram -- Load Annotation



UML Sequence Diagram -- Add Label



UML Sequence Diagram -- Get Label

The third major component of the backend system is the tests located in the /tests directory. Each test file tests different sets of user stories. For example, test_image.py tests the integrity of the image orm model

in SQLAlchemy. The test_label_endpoint.py tests all user stories regarding labels (i.e. loading and adding labels).

The fourth and final component of the backend system is the configuration files and scripts. These are essentially the files to start the application, deploy the backend system to the Azure VM, containerize the application, and set the project's configuration settings (i.e. the IP address of the VM and database, type of database, database username and password). These files are located in the primary directory and /scripts folder.

WebAssembly Development

Originally this project was aiming to produce both a desktop version and a WebAssembly (WASM) version that could be run remotely via a web page, from the same codebase. WebAssembly for Qt is a very new technology and has some notable restrictions. Many of these have to do with security restrictions applied by browsers to apps running in the JavaScript sandbox. Several of these restrictions were known ahead of time and workarounds were created, while others were not discovered until further into the project.

The primary restriction that we knew about ahead of time was that WASM apps do not have direct access to the local file system. The default Qt deployment creates a temporary file system in memory, but that is not permanent. A workaround for this was to use the browser's native JavaScript open file dialog to open projects and images, and use the browser's download feature to save outputs, but only as s single zip file.

The biggest restriction was when the networking interface was incorporated. Due to WASM running in a JavaScript sandbox, security restrictions prevent it from having a DNS lookup, or contacting the backend on any other machine that the one hosting the WASM web page. Both of these are doable (use static IP addresses, and be sure to host the WASM pages on the same server used for backend work). However, even though the documentation for Qt WASM stated that the Networking package would work, testing and reading of the code indicates that it does not. Specifically, no payload is attached to the outgoing POST, only the header was sent. We were able to hack the Qt code to make it work, but have not been fully able to test all functionality. Thus the WASM front end is given as-is, with plenty of future work to do. Future development of this code will require the cutting edge Qt 5.13 and the necessary patches.

Reflections and Lessons Learned

- Zonglin: I have learned a lot about the technologies, and development process. I learned about the common technologies used on the backend, such as Flask, Sqlalchemy and gained some experience of setting up a production server (install and config DB, configure a web server such

as Nginx etc.). Furthermore, I got a chance to explore some more advanced features of Github such as Github Project and task automation. Working in a team definitely helped me to learn a more efficient software development process, such as Agile. I also have learned techniques that can facilitate communications, such as various UML diagrams and got some experience with planning.

- Nishanth: I learned a lot from this project on not only new technologies and skills (e.g. Flask, Git, TravisCI) in software engineering but more about how software systems are built efficiently in industry and the best practices in doing so. The emphasis on Agile development throughout the project helped me understand how to plan out and implement projects quickly through Agile principles: incrementally working solutions after each iteration, test-driven development, and user stories and requirements. I was also able to improve my code quality, how to navigate through code review processes, and write high-quality tests through this process. Most importantly, I learned how to efficiently work in a software engineering team via many aspects such as planning meetings, discussing and agreeing on system designs, and writing conflict-free and standardized code to a common repository.
- Yu: I have learned a lot about the development process as well as the technologies throughout the whole process. While I had some previous experience with backend, during developing the InstaDam backend, I learned a lot of new things from my teammates and developed a deeper understanding of how certain backend frameworks/libraries work (e.g. SqlAlchemy ORM, Flask). Setting up CD & dockerize the app and environment was also a challenging task and luckily we figured things out and I also learned a lot on Docker and how CI-CD pipeline works. Most importantly, I had a taste of utilizing different development processes in a software engineering team. I have familiarized with Agile principles and process, which I think will benefit me a lot when I enter the industry later.
- Wanxian: This is one of the few team project experience for me over these years, and it is a valuable one. I think one of the most important things that I learned from this experience is how to work with teammates on Github via its team-working features such as project, commenting, assigning tasks, etc. It is also interesting to know about the tools that we have used, such as Flask, PostgreSQL and SQLAlchemy. Continuous testing/integration was completely new to me before this project. Now I agree that they are very important aspects of software development as they ensure the quality of deliverables to a large degree.
- Friedel: While I had some previous experience with Qt it was mostly in Python. There was a bit of a learning curve when it came to using it in C++ as some of the functionality is handled in a different way. Additionally, I learned that the information and claims from the developers (even in their bug fixes) for the cutting edge Qt WASM was sometimes false, even misleading. Simple tests showed me that the claimed functionality would never work as coded. This will lead me to

be more careful when examining new technologies in the future, just because they say it works doesn't mean it actually does.

- Fouad: This experience has been very informative for me on many aspects. Technically, I learned how to use C++ and how to design graphical interfaces with Qt. In terms of software engineering, it was my first time building a software together with other people, and it was an eye opener in terms of appreciating the importance of designing the software development process. I learned that communication is key, and the earlier potential risks are identified and concepts are discussed and agreed upon, the more productive the team is. Additionally, I learned about CI platforms such as Travis and i strengthened my understanding of git. Lastly, in terms of coding, i learned to be more generous and more careful in my naming conventions and in my documentation and to modularize as much as possible.
- Prahlad: This project was the first experience for me of building a new software from scratch. I had previously worked on teams and agile systems for existing software. I took for granted how well set up the process was when I realized how difficult implementing our own software process was. Design decisions were not easy to come up with and took many meetings to decide on. I knew and had some experience with C++/Qt so I didn't feel lost on the technical aspect of the project, but I do think I was lacking in some of the agile parts of the process. Communication is also something I need to improve at. I believe the front-end team did not communicate with the back-end tell enough until towards the end of the project, which led to a lot of confusion. In the future, I plan to be more attentive to design decisions and coding style, as well as reviewing the code of others more often to see what changes are being added so I am not surprised when trying to make changes.
- Vedhus: Had a great experience working with our awesome team in building software from scratch. Learnt a lot about software development and slowly understood the true importance of following an agile process. I learnt that communication between team members cannot be taken for granted and after technical knowledge, communication is perhaps the most important aspect of building a working software product. Further, I became well versed in Qt and OpenCV for C++ and familiar with using git. I also discovered the various issues that may crop up if a proper software engineering process is not followed where changes to one part of the code can break other parts.