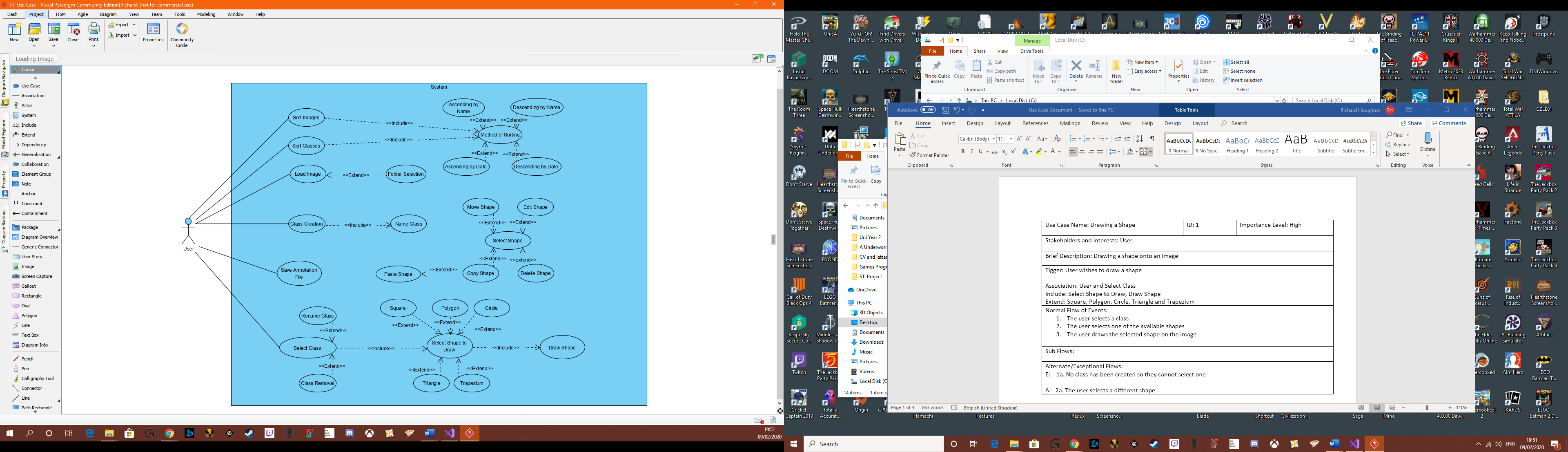
**Software Architecture**

**Use Case Diagram:**

|  |  |  |
| --- | --- | --- |
| Use Case Name: Drawing a Shape | ID: 1 | Importance Level: High |
| Stakeholders and interests: User | | |
| Brief Description: Drawing a shape onto an image | | |
| Tigger: User wishes to draw a shape | | |
| Association: User and Select Class  Include: Select Shape to Draw, Draw Shape  Extend: Square, Polygon, Circle, Triangle and Trapezium | | |
| Normal Flow of Events:   1. The user selects a class 2. The user selects one of the available shapes 3. The user draws the selected shape on the image | | |
| Sub Flows: | | |
| Alternate/Exceptional Flows:  E: 1a. No class has been created so they cannot select one  A: 2a. The user selects a different shape | | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: Making Changes to a Class | ID: 2 | Importance Level: High |
| Stakeholders and interests: User | | |
| Brief Description: Making changes to an existing class | | |
| Tigger: User wishes to change an existing class | | |
| Association: User, Select Class  Include:  Extend: Rename Class, Class Removal | | |
| Normal Flow of Events:   1. The user selects a class 2. The class gets renamed 3. All shapes made using that class are updated with its new name | | |
| Sub Flows:  S-1:   1. The user selects a class 2. The user deletes the class 3. The user is asked if they are sure about deleting the class 4. The class is deleted and all corresponding shapes along with it | | |
| Alternate/Exceptional Flows:  E: 1a. No class has been created so they cannot select one | | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: Editing or Otherwise Manipulation a Shape | ID: 3 | Importance Level: High |
| Stakeholders and interests: User | | |
| Brief Description: Editing an existing shape | | |
| Tigger: User wishes to edit an existing shape | | |
| Association: User, Select Shape  Include:  Extend: Move Shape, Edit Shape, Delete Shape, Copy Shape, Paste Shape | | |
| Normal Flow of Events:   1. The user selects a shape 2. The user moves the shape around the image | | |
| Sub Flows:  S-1:   1. The user selects a shape 2. The user edits the size of the shape   S-2:   1. The user selects a shape 2. The user copies the shape 3. The user then pastes an identical copy of the shape onto an image   S-3:   1. The user selects a shape 2. The user deletes the shape 3. The shape is removed from the image | | |
| Alternate/Exceptional Flows:  E: 1a. No shape has been created so they cannot select one | | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: Sorting Lists | ID: 4 | Importance Level: High |
| Stakeholders and interests: User | | |
| Brief Description: Sorting the images and classes by various methods | | |
| Tigger: User wishes to sort one of the lists | | |
| Association: User, Sort Image, Sort Classes  Include: Method of Sorting  Extend: Ascending by Name, Descending by Name, Ascending by Date, Descending by Date | | |
| Normal Flow of Events:   1. User selects what they wish to sort 2. The user then selects the method of sorting they wish to use 3. The list is then sorted by the parameters of the sorting method | | |
| Sub Flows: | | |
| Alternate/Exceptional Flows:  E: 3a. The lists are empty so cannot be sorted | | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: Creating a Class | ID: 5 | Importance Level: High |
| Stakeholders and interests: User | | |
| Brief Description: Creating a class | | |
| Tigger: The user wishes to create a class | | |
| Association: User, Class Creation  Include: Name Class  Extend: | | |
| Normal Flow of Events:   1. The user selects create a class 2. A prompted comes up to name the newly created class 3. A new class is created | | |
| Sub Flows: | | |
| Alternate/Exceptional Flows:  E: 2a. The user doesn’t provide a name, so they are prompted once more to name the class | | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: Loading Images | ID: 6 | Importance Level: High |
| Stakeholders and interests: User | | |
| Brief Description: The process of loading images into the program | | |
| Tigger: The user wishes to load images into the program | | |
| Association: User, Load Image  Include:  Extend: Folder Selection | | |
| Normal Flow of Events:   1. The user selects load image 2. A file browser opens 3. They select an image to be loaded into the program 4. The image is loaded into the program | | |
| Sub Flows:  S-1:   1. The user selects load image 2. A file browser opens 3. They select a file of images to load 4. All the selected images are loaded into the program | | |
| Alternate/Exceptional Flows:  E: 3a. The user does not select anything and are prompted to select an image to load  3b. The user selects an incompatible file and are notified | | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: Saving the Annotation File | ID: 7 | Importance Level: High |
| Stakeholders and interests: User | | |
| Brief Description: The process of saving the annotation file | | |
| Tigger: The user wishes to save their work | | |
| Association: User, Save Annotation File  Include:  Extend: | | |
| Normal Flow of Events:   1. The user clicks on the save button 2. An annotation file is created 3. A file browser appears asking the user to name the file and where they wish to save it to 4. The file is saved | | |
| Sub Flows: | | |
| Alternate/Exceptional Flows:  E: 2a. Not all the components have been added in order to create an annotation file and as such one q won’t be created and the user will be notified  3a. The user does not name the file and if prompted to do so | | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: | ID: 8 | Importance Level: High |
| Stakeholders and interests: User | | |
| Brief Description: | | |
| Tigger: | | |
| Association: Include:  Extend: | | |
| Normal Flow of Events: | | |
| Sub Flows: | | |
| Alternate/Exceptional Flows: | | |

**Class Diagram:**

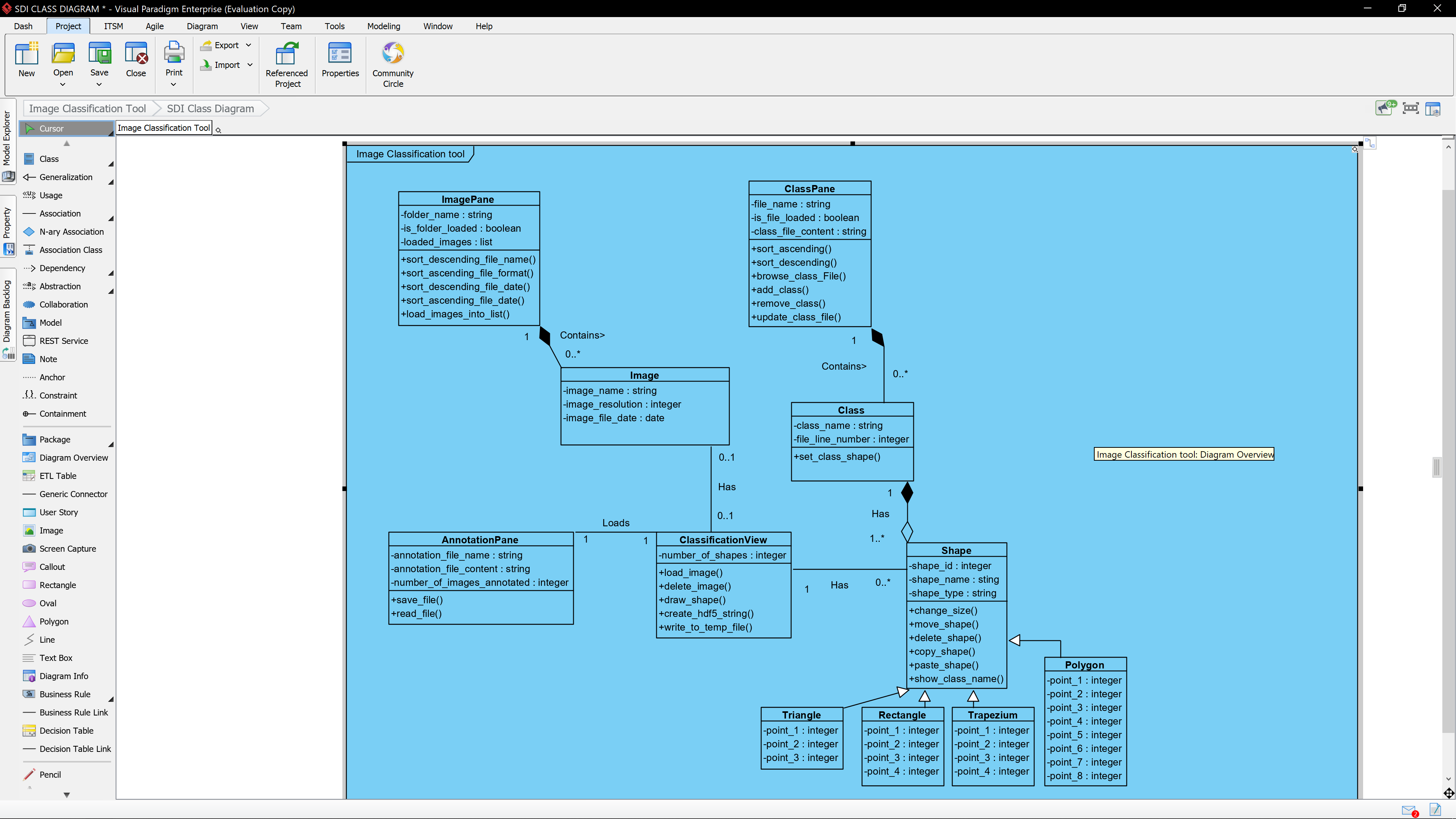


Image Class – Image Pane: an image must be included in at least one instance of an ImagePane, until an Image is loaded within the ImagePane an instance of the Image class won’t exist thus making it a composition.

ImagePane Class – Image Class: an ImagePane will contain zero or many instances of an image, depending on how many are loaded in.

Image Class – ClassificationView: An image can be loaded into a single instance of ClassificationView or not.

ClassificationView – Image Class: The ClassificationView can have a single image loaded into it or none.

ClassificationView – Shape Class: The ClassificationView can reference several instances of class Shape.

Shape Class – ClassificationView: Each instance of the shape class must be loaded into a single instance of ClassificationView.

Shape Class – Class Class: Each instance of a shape must be related to a single instance of a Class.

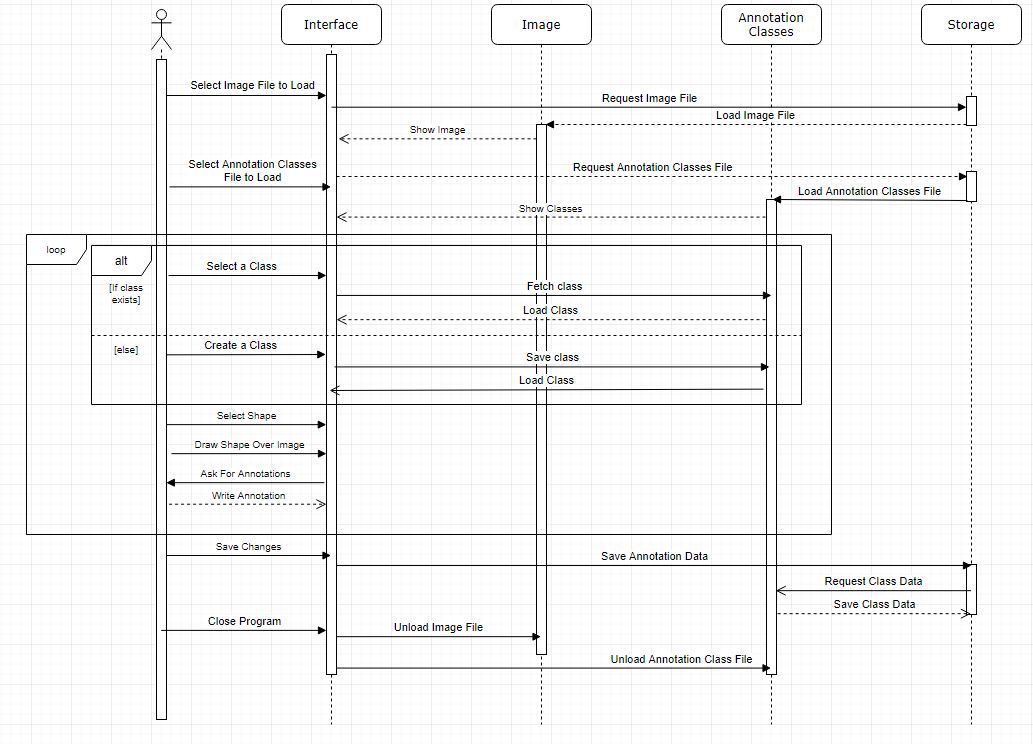
Class Class – Shape Class: Each Instance of the Class Class has between 0 to many shapes which will be linked to it. Relationship Class – Shape is aggregation as a class can exist without the need for a shape to exist.

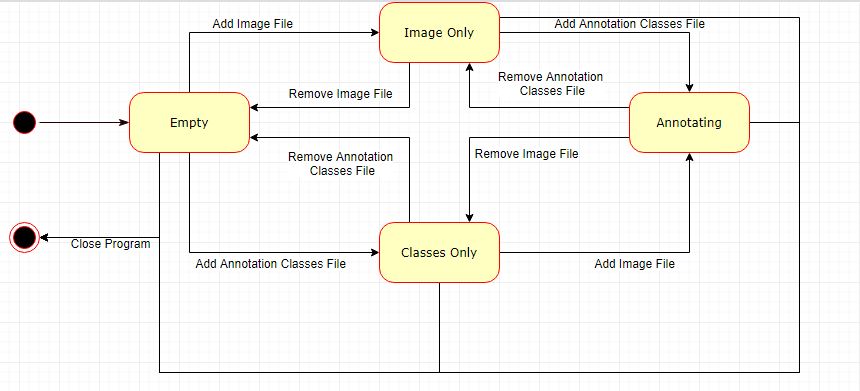
Class – ClassPane: A Class Class must be included in at least one instance of a ClassPane, until a ClassFile is loaded in by an instance of ClassPane a Class instance wont exists thus making this a composition.

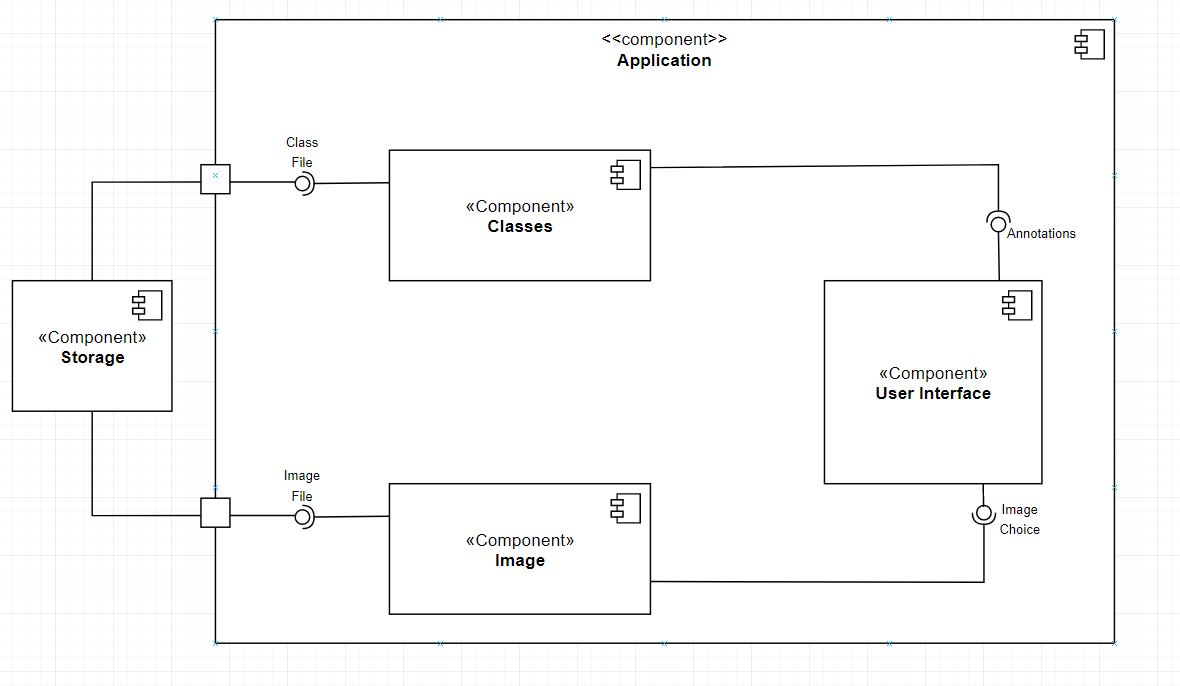
ClassPane – Class: ClassPane will contain zero or many instances of Class.

AnnotationPane – ClassificationView: One to one relationship as each instance of AnnotationPane will load an instance of ClassificationView.

ClassificationView – AnnotationPane: Single instance of ClassificationView relates to a single AnnotationPane.

**Sequence Diagram:**

**State Diagram:**

**Component Diagram:**

**Tools, Libraries and Frameworks:**

**Qt**

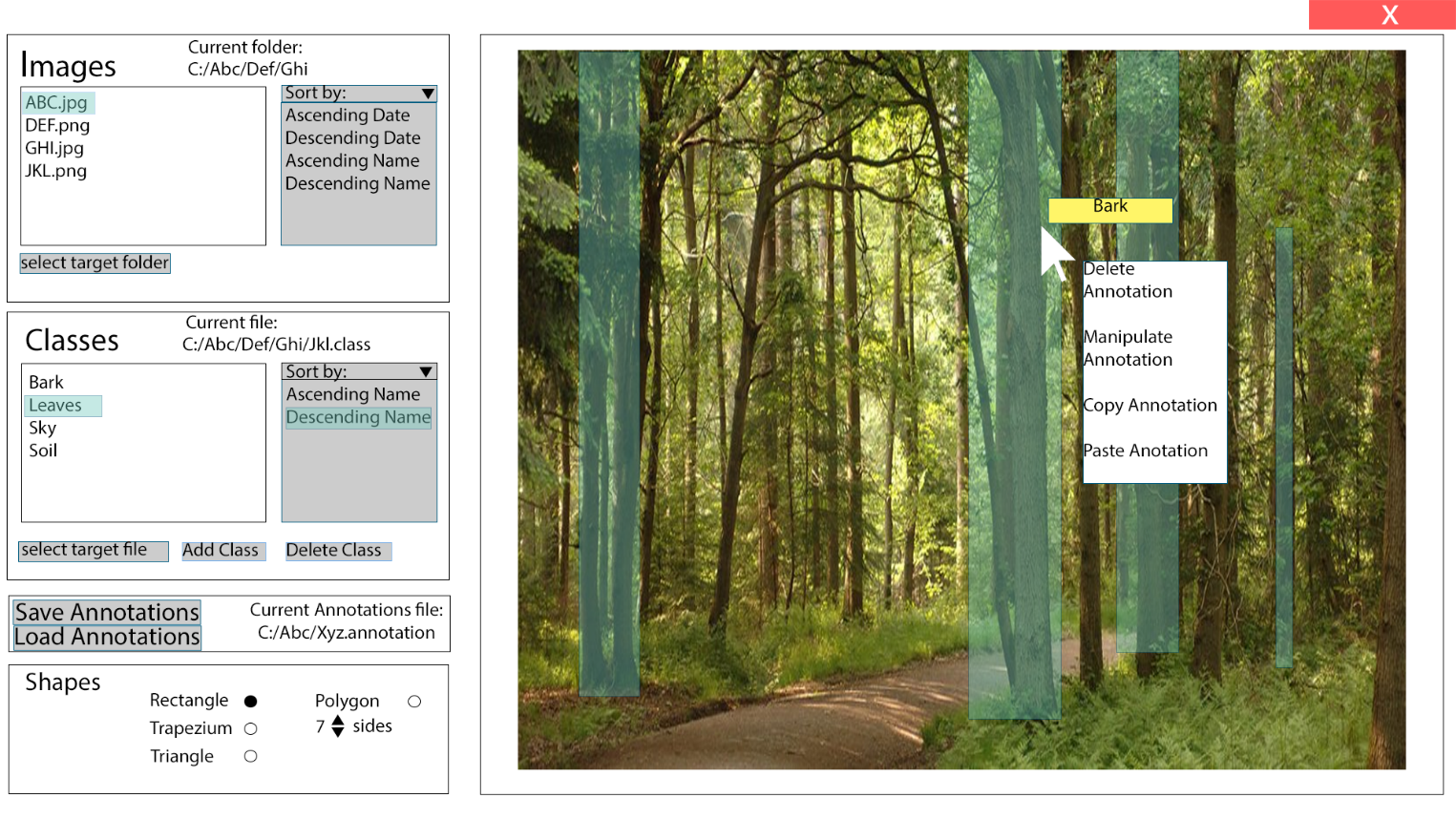
Qt will be the graphical user interface (GUI) framework that will be used for this project. Qt was chosen due to its easy of use and extensive list of libraries contained within it.

There are a number of example applications with corresponding documentation meaning learning how to use it will be a straightforward process for the team and as no members have any previous experience with any C++ GUI frameworks so an easy and quick to learn one is highly beneficial.

Having been originally released in 1995 with continues development since Qt has an extensive array of libraries which will easily enable the programming of the GUI necessary for this project.

**CImg**

For image processing the CImg library will be used. It fulfils all the needs of this project with its library that supports the loading and saving of images, the ability to list the images and displaying images. CImg is a compact library the that will only take up 1MB of storage space and is well optimises so as to not take up any unneeded processing space. It is also not too complex so it will not take much time for the team to understand how to implement it and use it with in the code of the project.

**GUI Mock-up**