Introduction and User Stories

The goal of this system is to store and organize different types of resources that users come across in their daily lives. Resources exist within categories that are relevant to the resource for easy access. Categories can contain subcategories for more specific resource storing and will be implemented with a tree like structure.

Take for example a computer science professor, Anna Adam, who always learns new things about the industry that she can teach her students, but always loses track of all the useful resources she finds. She can use this application to organize and can have categories for teaching and research in her professional life and also things like recipes for her personal life. Within the teaching category she can have all of the courses that she teaches and within each class she can store articles that she finds relevant to the content of each. The structure of the application will be exactly like a file tree and will provide frameworks for storing all the different types of resources in a way that is relevant to the resource. For example a video resource object can contain a link to the video and play the video within the application.

In another example, Fatma Akgül, a Software Engineer Job Applicant, gained so much useful information throughout the job search process but she can never find all these resources. She has taken multiple networking calls, read a ton of useful articles about applications and resumes, and has even watched TikToks that give good examples of good responses to interview questions. She needs a way to organize all of these resources in a way that is easy to use and track resources. Our application can help solve these problems.

Object Oriented Design

As stated in the introduction, this structure of the resource storing mechanism will be a tree similar to a file storing structure. The user interface will allow for easy traversal of the tree structure in a visual way for quick storing and retrieval of stored resources. At the highest level the graphical user interface will consist of multiple views for every different type of screen that will be displayed. There will be a separate controller for each view and a viewSwitcher class will provide functionality for easy view switching with one single method call. One instance of a model will exist within the program and it will get passed between all of the different controllers where the user can modify the model via the GUI. The model will interact with a CategoryNode class and a resource class and the CRC cards for this design are shown below.

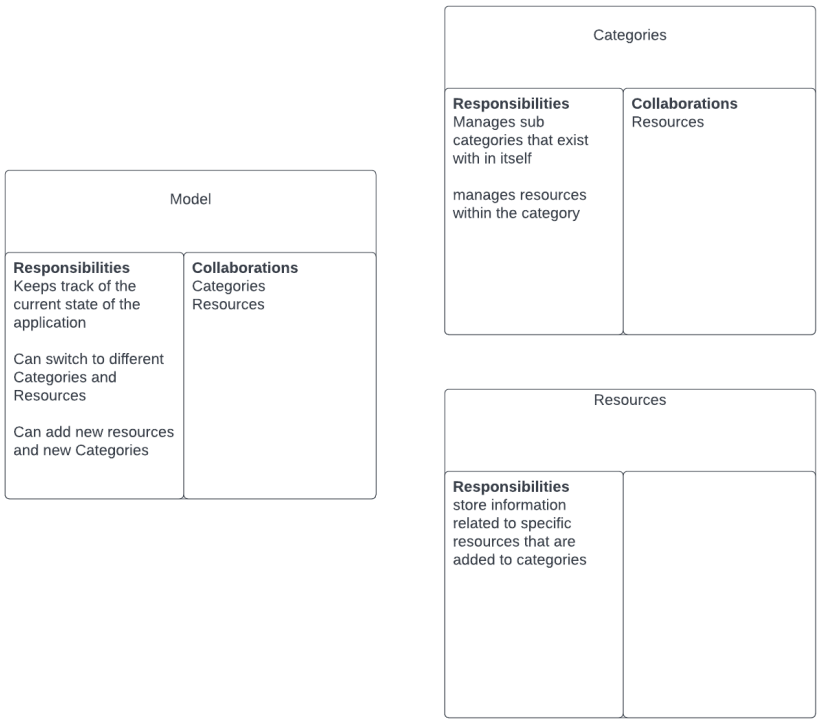

Figure 1 CRC Cards for the application design

A CategoryNode class will allow for a user to create categories that can store subcategories and resources. This will lead to a tree structure where a root CategoryNode will be the initial starting place for expanding the tree further. The model will contain this root and provide useful methods for easy tree additions, deletions, and traversal. The Resource class will be the lowest level object and will store the information that the user wishes to retrieve at a later time. The UML class diagram for the model is shown below.

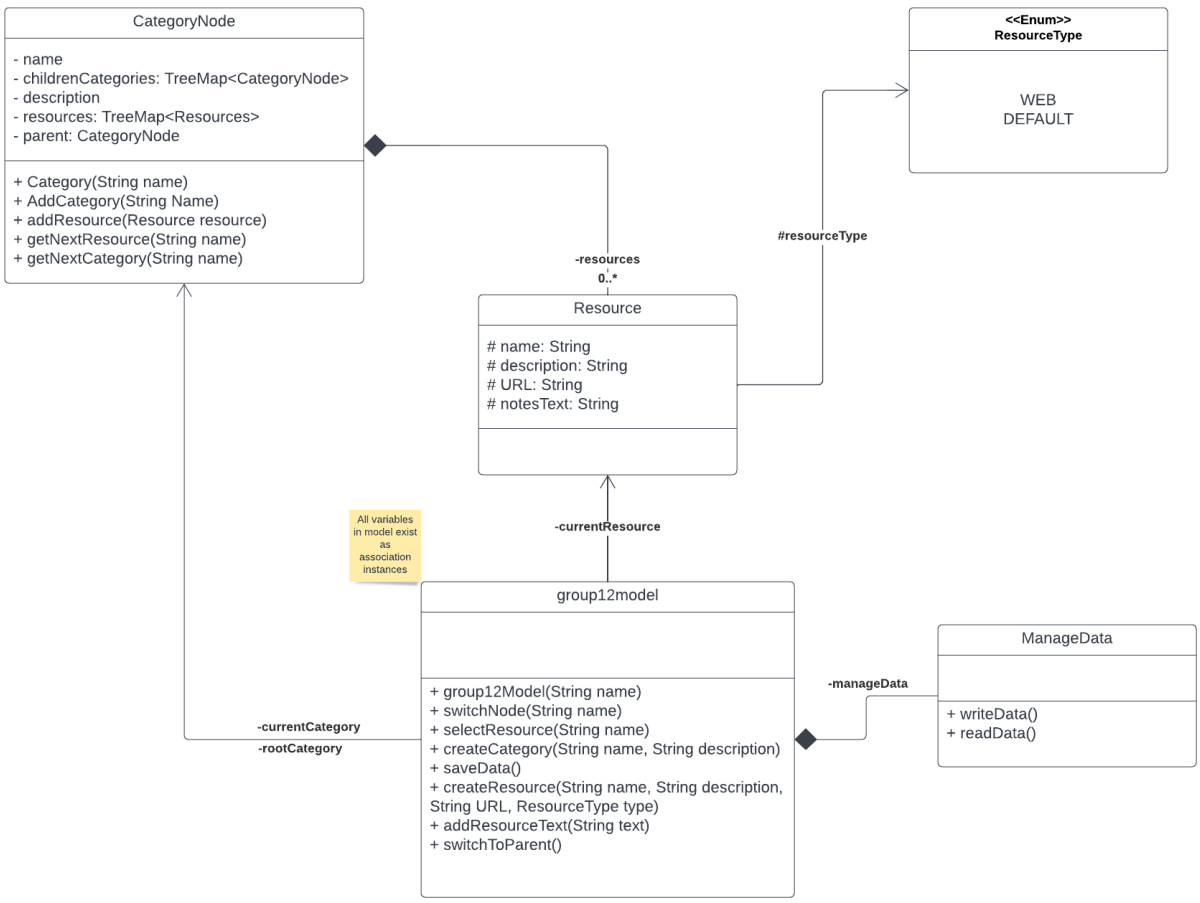

Figure 2 UML Diagram of the model

There will be different kinds of resources that can have different functionality based on the resource. For the initial design, two resource types will exist and that is a web resource and a

default resource. The web resource will display the website within the application and a default resource will allow for easy storing of user entered text. The users in the aforementioned user stories can keep a text based resource for themselves in a useful category or they can save links to important websites that they would like to revisit easily. The model functionality for both of these resources is the same therefore one resource class is needed for now, however the model is set up in a way that will allow for polymorphism of resources if future resources classes are derived from the parent with new functionality. For now, the difference in resources only exists within the controllers and the view therefore an enumeration can be used to differentiate types of resources.

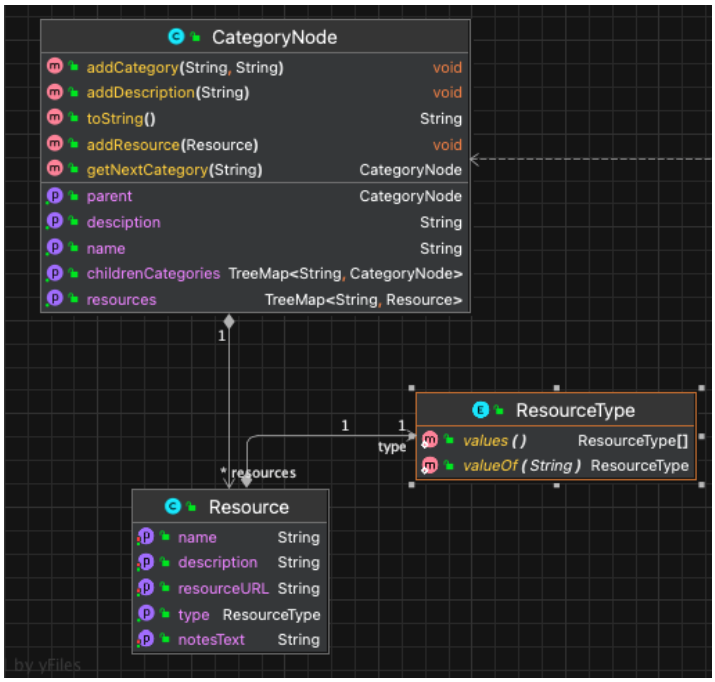

Figure 3 UML Class diagram showing relationship between CategoryNode and Resource

The final object in the above diagram is the ManageData object which is responsible for storing and retrieving data. This object is created by the model and has functionality for serializing the model to a .bin file. This allows for user data to be saved whenever new data is added and retrieved when the application is initially launched.

The final important design feature is the structure of controller classes to connect the many different views to the multiple and update the model/view based on user changes. The views are created in scene builder generating a .fxml file that is linked to its respective controller class. When a view is displayed via a .fxml file, a controller object is created automatically using a default constructor. To display a view, the relevant objects from the model need to be displayed but with the .fxml auto generating a controller, there is nowhere to pass the controller the current instance of the model. Therefore two new methods need to exist in every controller the first being a setModel() method and the second an initController() method to initialize the view with the state of the model. This is done with an interface that each controller implements that requires these two methods. The UML case diagram for the controller classes is shown below.

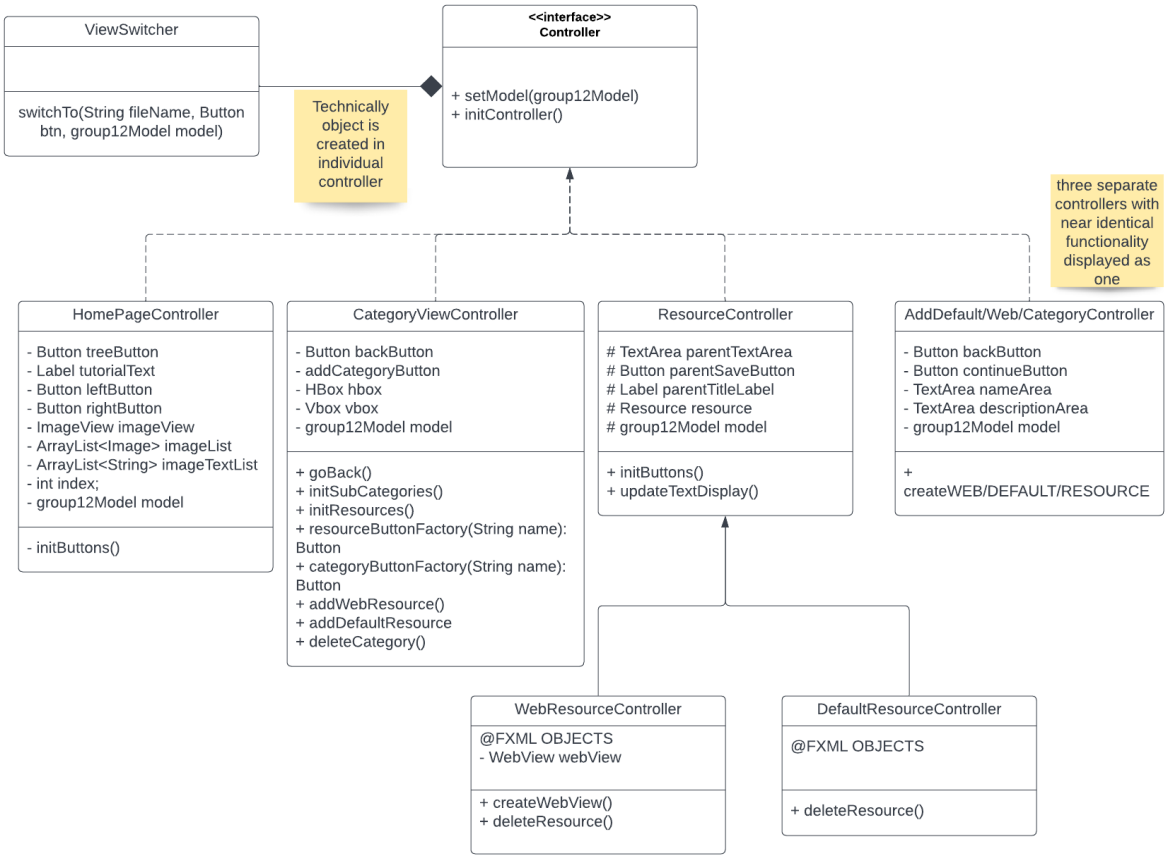

Figure 4 UML Controller Class Diagram

The separate ViewSwitcher class shown in the model handles all responsibilities for changing views. Given the location of the next .fxml file, current instance of the model, and the location of the object that called for a new view, the ViewSwitcher can call all of the necessary methods to change the current view. For example, the user is able to click the add new category button, get redirected to a screen to input the name of the new category and a description, and return to the category view with the new category being displayed. Additionally, different Resource types have different views but controllers for these views have shared functionality. Therefore an abstract Resource class contains methods for shared resource functionality. Different types of resources have controllers derived from this abstract class to handle functionality specific to each resource type.

In conclusion, Fatma and Anna can easily organize their different areas of interest in a way that is easy to view and navigate. They can seamlessly switch from screen to screen adding new resources they wish to come back to. When that time comes, they can quickly open the application, travel down the easy to visualize tree until they reach what they are looking for.

Sources

Serialization: https://www.geeksforgeeks.org/serialization-in-java/

WebView: https://jenkov.com/tutorials/javafx/webview.html

.fxml:https://stackoverflow.com/questions/34785417/javafx-fxml-controller-constructor-vs-initialize-method

Slideshow:https://stackoverflow.com/questions/43318958/how-to-create-a-javafx-image-slider-using-the-imageview