Code Review of The Software Development Project:
"File Sharing Desktop Application"

Course Title: Software Development Project
Course ID: CSE-3106

Project By:
Gayotri Gope Toma
Student ID: 210212
Jannatul Ferdous Shova
Student ID: 210228

Reviewed By:
Arnob Chakrobortty
Student ID: 210205
Samia Khanom Asa
Student ID: 210222

Submitted To:
Amit Kumar Mondol
Assistant Professor
Computer Science & Engineering Discipline
Khulna University,
Khulna.

Introduction:

This code review evaluates the **"File Sharing Desktop Application".**
The review identifies areas for improvement, adherence to best practices, and suggestions for enhancing maintainability, security, and overall code quality. In this code review, bad smells of the code, architecture evaluation, modularity check, condition statements of the code & other related sections are evaluated.

Code Smells:

This code appears to be a GUI application for file transfer using Tkinter in Python. Here are some code smells and potential improvements:

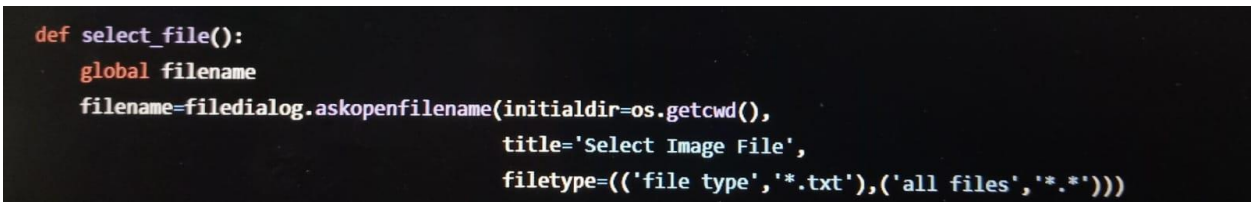
1. Organization and Structure:

- The entire application is in a single file. Breaking it into smaller modules or functions for better maintainability should be considered.

2. Global Variables:

- Using global variables should be avoided. They make code harder to understand and maintain.

- In the Send function, filename is declared as a global variable. It's better to pass this as a parameter to functions that need it.



3. Magic Numbers and Hardcoded Values:

- The port number 8080 is used in multiple places. Defining it as a constant at the beginning of the script will be better.

```
def sender():
    s=socket.socket()
    host=socket.gethostname()
    port=8080
    s.bind((host,port))
    s.listen(1)
    print(host)
    print('waiting for any incoming connections.... ')
    conn,addr=s.accept()
    file=open(filename,'rb')
    file_data=file.read(1024)
    conn.send(file_data)
    print("Data has been transmitted successfully")
```

```
s=socket.socket()
port=8080
s.connect((ID,port))
file=open(filename1,'wb')
file_data=s.recv(1024)
file.write(file_data)
file.close()
print("File has been received successfully")
```

- There are several hardcoded values for window dimensions and positions. defining them as constants for better adaptability will be better.

4. Code Duplication:

- There's some duplication of code in the Send and Receive functions, especially for creating and configuring new windows. Refactoring common functionality into separate functions will be better.

```
def Receive():
    main=Toplevel(root)
    main.title("Receive")
    main.geometry('450x560+500+200')
    main.configure(bg="#f4fdfe")
    main.resizable(False, False)
```

```
def Send():
    window=Toplevel(root)
    window.title("Send")
    window.geometry('450x560+500+200')
    window.configure(bg="#f4fdfe")
    window.resizable(False,False)
```

5. Error Handling:

- Error handling is minimal.

- Adding try-except blocks to handle potential errors, especially around file operations and network connections will be better.

6. Naming Conventions:

- Using descriptive variable and function names will be better. For instance, Sbackground, Mbackground, and Hbackground are not very informative. Using names that clearly describe their purpose will be perfect.

```
Sbackground=PhotoImage(file="images/sender.png")
Label(window,image=Sbackground).place(x=-2,y=0)

Mbackground=PhotoImage(file="images/id.png")
Label(window,image=Mbackground,bg=("#f4fdfe")).place(x=100,y=260)
```

7. Comments:

- Adding comments to explain complex logic or non-obvious functionality will be more useful. Comments will make the code more understandable for other developers.

8. UI Design:

- The UI design could be improved for better user experience. For example, adding labels to input fields to provide context can be considered.

- Ensuring consistent use of fonts, colors, and spacing throughout the application should be implemented.

9. Resource Management:

- Proper resource management should be ensured, especially for file handles. Files should be closed even if an error occurs during file transfer..

10. Separation of Concerns:

- The code mixes UI logic with network communication. Separating these concerns into different modules or classes for better maintainability should be considered.

11. Testing:

- Unit tests and integration tests should be considered to ensure the functionality works as expected, especially for network operations.

12. Security:

- The code doesn't address potential security concerns, such as input validation and protection against malicious inputs. Ensuring input from users for sanitizing and validating will be better.

By addressing these code smells, the readability, maintainability, and reliability of the code can be improved.

Proposed Architecture Evaluation:

The provided code does not strictly follow the Model-View-Controller (MVC) architecture. MVC is a design pattern commonly used in software engineering for separating concerns within an application. It divides an application into three interconnected components:

1. **Model:** Responsible for managing the data, logic, and rules of the application.
2. **View:** Represents the presentation layer, responsible for displaying data to the user and handling user input.
3. **Controller:** Acts as an intermediary between the Model and the View, responsible for interpreting user inputs and invoking appropriate actions on the Model and View.

In the provided code:

- There is no clear separation of concerns into Model, View, and Controller components.
- The code primarily deals with the user interface (View) and some networking functionalities. There is no distinct Model component handling data and logic separately.
- Functions like **Send()** and **Receive()** perform actions directly related to the user interface and networking operations, but they do not clearly represent controllers in the MVC sense.

To refactor the code to follow MVC architecture:

1. **Separate concerns:** We have to identify the data management and business logic (Model), user interface (View), and user input handling (Controller) parts of the application.
2. **Refactor code:** We have to organize the code into separate modules or classes for Model, View, and Controller components.
3. **Implement communication:** Ensure that the components communicate appropriately, following the MVC pattern's guidelines.

Modularity Check:

This code demonstrates some modularity by separating functionalities into different functions and organizing them under appropriate sections. This code has separate functions for sending and receiving files (Send() and Receive()), which is good for modularity. Each function handles a specific action, making the code easier to understand and maintain each action (sending and receiving) is encapsulated within its own function, which promotes modularity by isolating different behaviors from each other. Functions like select_file() and receiver() can be reused if similar

functionality is needed elsewhere in the program. This promotes modularity and reduces code duplication.

If/else Condition to Switch statement:

There is no built-in switch case statement in python. As a result, there is no other way to implement the conditions required in the code other than using if/else statement.