🧱 Overall Structure

Your app will have two main sections:

1. **Embed Mode** – Where the user hides an encrypted message inside an image.
2. **Extract Mode** – Where the user provides a stego-image and a key to retrieve the hidden message.

Each mode has its own flow but shares common features like image upload and key input.

🎨 User Interface Components

1. Home/Welcome Screen

- App title (e.g., *“StegoCrypt”* or *“Unveil GUI”*).
- Brief description: “Encrypt and hide messages inside images using steganography.”
- Two big buttons:
    - "Hide a Message"
    - "Reveal a Message"

2. Embed Mode (Hide a Message)

🖼️ *Image Upload Section*

- Upload button to select an image from your device.
- Accepted formats: `.png`, `.jpg`, `.webp`.
- On upload:
    - Show a small preview of the image.
    - Display image metadata: resolution, format, color mode.
    - Check image size to make sure it’s large enough for embedding.
    - If not, display an error.

📄 *Message Input Section*

- Option 1: Textbox where user can type a message directly.
- Option 2: File upload button for a `.txt` file.
- Optionally display the first few lines if it's a large file.
- Validation:
    - Non-empty message.
    - Max size based on the image's pixel capacity.

🔑 *Encryption Key Section*

- Text field to enter a numeric key.
- Tooltip: "This key is used to encrypt and decrypt your message. Keep it safe!"
- Option to generate a random key and display it.
- Option to export/save the key as a `.txt` file.

🔓 *Encrypt & Embed Button*

- When clicked:
    - Encrypt the message using your Caesar shift logic.
    - Convert encrypted text to binary.
    - Embed it into the image using your LSB algorithm.
    - Show progress bar or spinner during processing.

🖼️ *Output Section*

- Display preview of the modified image (stego-image).
- Show confirmation message: "Your message has been securely embedded."
- Provide a download button to save the stego-image locally.
- Optional: show how many bits were used, success/failure logs.

3. Extract Mode (Reveal a Message)

🖼️ *Stego-Image Upload Section*

- Upload button for a previously saved stego-image.
- Image is displayed in a preview window.
- You validate that it's an acceptable format and compatible size.

🔑 *Decryption Key Input*

- Field for user to enter the numeric key they were originally given.
- Optional: Upload a key file.
- Validate that the key is a valid integer and within range.

🧠 *Extraction & Decryption*

- Button: “Reveal Hidden Message”
- What happens under the hood:
    - Your tool reads the LSB bits from the image.
    - It decodes the binary stream into text.
    - It decrypts the text using the provided key.
    - Message is shown in a textbox (scrollable).
- Error messages shown if:
    - No message was found.
    - The key is wrong or leads to garbage output.
    - The image wasn’t created using this tool.

🛡️ Optional Security Features

- **Checksum or Signature**: Embed a hidden “signature” so your program knows whether an image was tampered with or not created by your app.
- **Error Handling Popups**: Modal windows for bad uploads, incorrect keys, corrupted images.
- **Logs Panel**: Scrollable text area showing what happened behind the scenes (useful for developers or debugging).

⚙️ Settings / Extras

- **Dark mode toggle**
- **Export logs**
- **Help/About section**
- **Credits / GitHub link / Documentation**

🧪 Test Cases to Prepare For

- Large messages that almost fill the image's capacity.
- Incorrect image format (e.g., BMP, TIFF).
- Non-RGB images (grayscale, CMYK).
- Corrupted or tampered stego-images.
- Users uploading a message and forgetting to input a key.
- Users trying to decrypt with the wrong key.