Product Design

Team : Colour Auto Correction - Team 46

**Team Members: Abhinav Raundhal, Ishan Gupta
Harsh Gupta, Deekshitha Yattapu, Sujal Deoda**

NOTE: The UML tool used for making the class diagrams Visual Paradigm
The UML tool used for making the sequence diagram is Star UML

Links to Diagrams:-

1. ClassDiagram.vpp://diagram/2pp0dHGAUAkG_wyX
2. SequenceDiagrams: https://iiitaphyd-my.sharepoint.com/:f:/g/personal/abhinav_raundhal_students_iiit_ac_in/EnkQtUw9jJxJqHtBfO43PRYBhUWd0PhUc5c9C1x241emeg?e=4mxBbv

Design Model

User	Class State ● Username: A unique identifier for the user, used during the login process. ● Password: A secret key used alongside the username to authenticate the user. Class Behavior. ● uploadPhotos(photoSet): Allows the user to upload a batch of photos for processing. The `photoSet` could be a collection of image files. ● reviewEdits(photo): Enables the user to view the edited version of an uploaded photo and compare it to the original. ● approveEdit(photo): Lets the user confirm that the edits made to a photo are satisfactory and final. ● requestAdjustment(photo, parameters): Allows the user to request further adjustments to a photo, specifying desired changes via `parameters`.
SystemAdmin	Class state: ● adminUsername: A unique identifier for the system administrator. ● adminPassword: A secret key for the system administrator's authentication. Class Behavior: ● manageDatabase(): Performs database management tasks such as backup, recovery, and ensuring data integrity. ● updateAlgorithms(): Updates the algorithms used for photo analysis and color correction to improve accuracy or efficiency.

PhotoAnalyzer	Class Behavior: ● analyzePhoto(photo): Analyzes an uploaded photo to determine characteristics such as brightness, contrast, and color balance. ● extractParameters(photo): Extracts specific parameters from an analyzed photo that are necessary for color correction.
Database	Class State: ● adjustmentPreset: A predefined set of adjustment parameters that can be applied to photos for quick color correction. ● connectionDetails: Information necessary to establish a connection to the database, such as host, port, database name, username, and password. Class Behavior: ● retrieveParameters():Fetches pre-computed adjustment parameters from the database for a new set of photos. ● storeParameters(adjustmentFactors): Saves adjustment factors calculated by the system into the database for future reference. ● storeEditedPhotos(photo): Saves the edited photos back into the database, preserving the changes made during the color correction process.
ColorCorrector	Class State: ● adjustmentPreset: A predefined set of adjustment parameters that can be applied to photos for quick color correction. Class Behavior: ● applyCorrection(photo, parameters): Applies color correction to a photo using specified `parameters` or a selected preset. ● retrievePreset(presetId): Fetches a specific adjustment preset from a collection based on `presetId`.

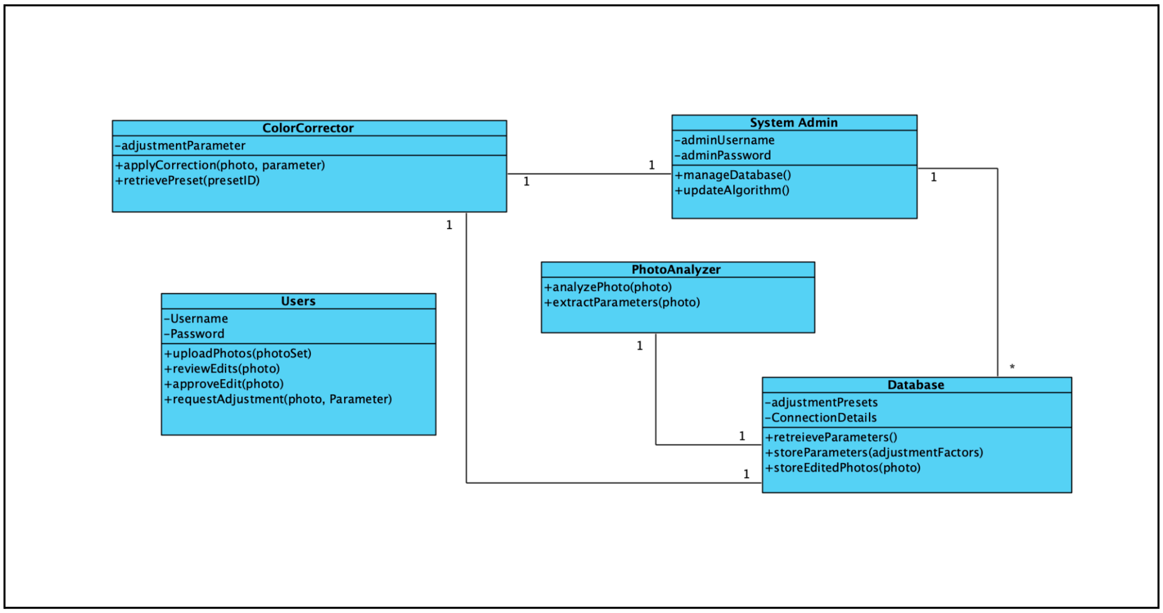

Fig 1. UML Class Diagram

These classes collectively support a system that enables professional photographers or photography studios to upload batches of photos for automated analysis, color correction, and review. System administrators manage the system's backend operations, ensuring smooth functioning and security.

Sequence Diagram(s)

The following are the details of the use cases used in the Sequence diagrams shown below.

Fig.2 depicts the use cases Upload Photos, Extract Parameters

Fig.3 depicts the use cases Apply Color correction

Fig.4 depicts the use cases Manual User Approval

Fig.5 depicts the use cases Personalized Color Correction

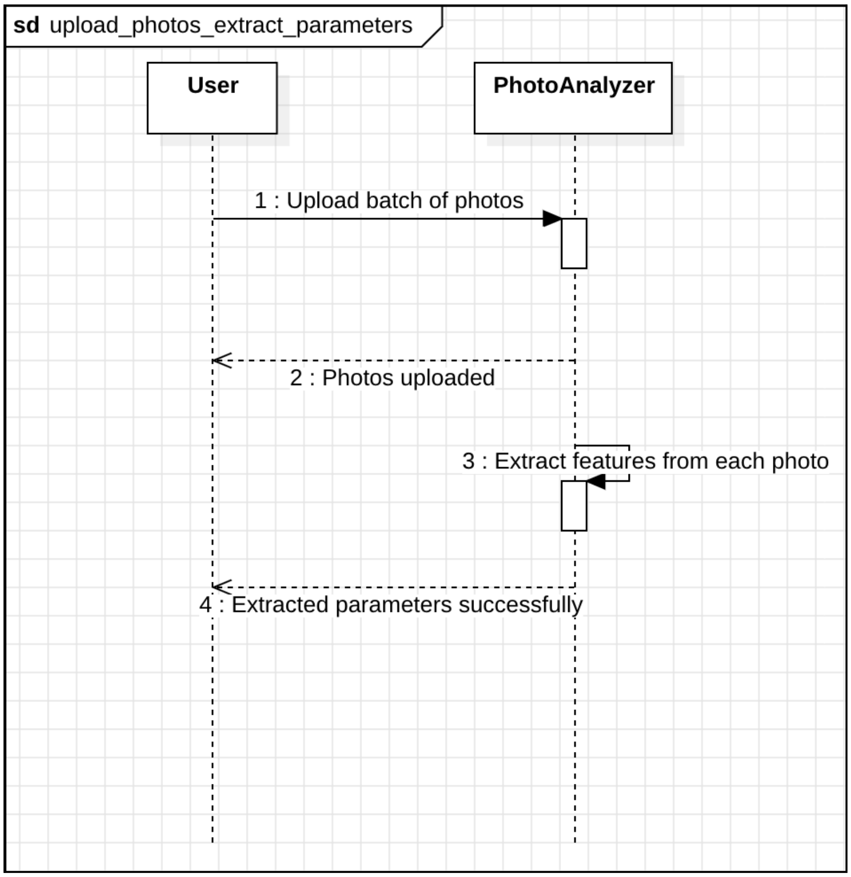
sd upload_photos_extract_parameters
User
PhotoAnalyzer
1 : Upload batch of photos
2 : Photos uploaded
3 : Extract features from each photo
4 : Extracted parameters successfully

Fig. 2

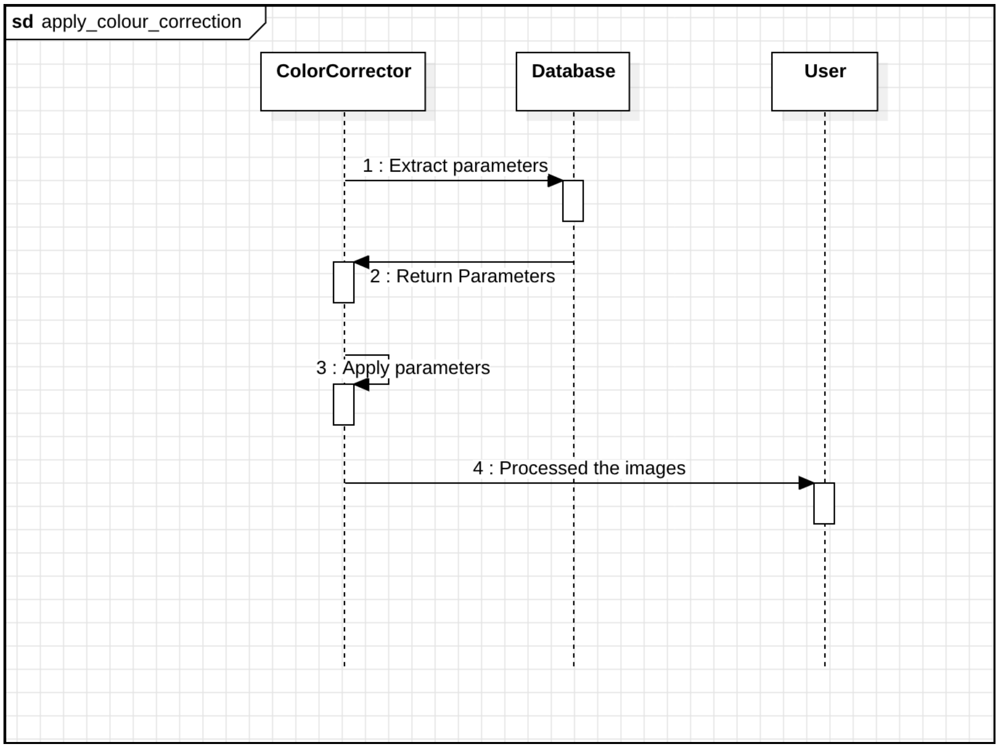
sd apply_colour_correction
ColorCorrector
Database
User
1 : Extract parameters
2 : Return Parameters
3 : Apply parameters
4 : Processed the images

Fig. 3

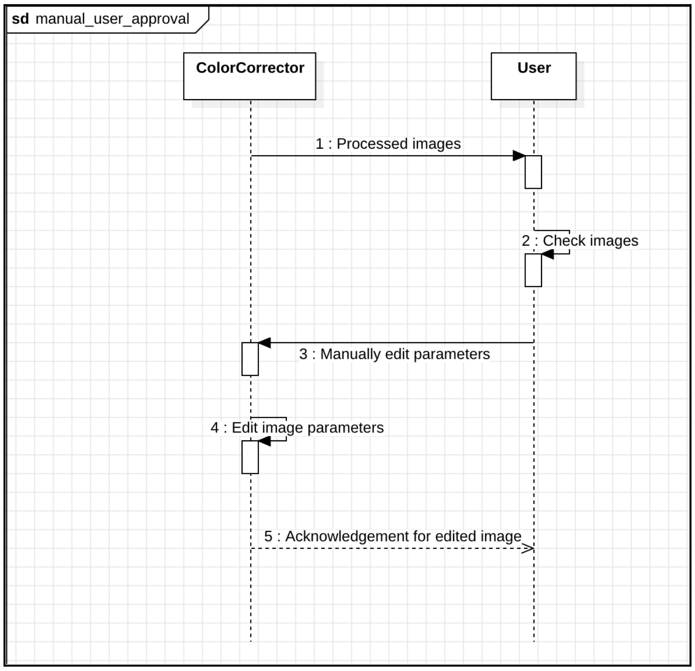
sd manual_user_approval
ColorCorrector
User
1 : Processed images
2 : Check images
3 : Manually edit parameters
4 : Edit image parameters
5 : Acknowledgement for edited image

Fig. 4

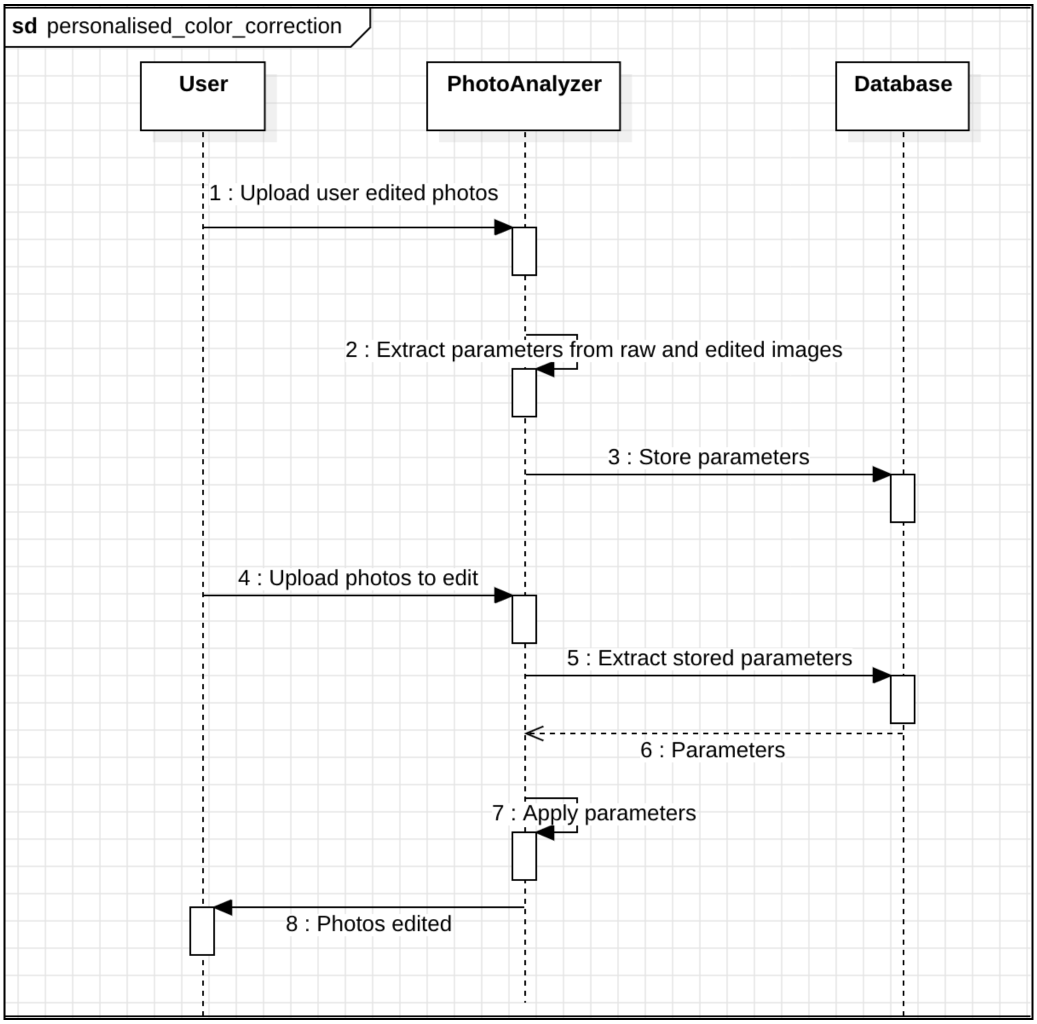
sd personalised_color_correction
User
PhotoAnalyzer
Database
1 : Upload user edited photos
2 : Extract parameters from raw and edited images
3 : Store parameters
4 : Upload photos to edit
5 : Extract stored parameters
6 : Parameters
7 : Apply parameters
8 : Photos edited

Fig. 5

Design Rationale

1. **User Photo Upload Interface:**For the User Photo Upload Interface, addressing the cumbersome initial photo upload process was critical. We deliberated between a Single Photo Upload, which, while straightforward to develop, was inefficient for users with multiple photos, and a Drag-and-Drop Batch Upload, which offered an intuitive and quick solution for uploading multiple photos despite potentially more complex coding requirements and browser compatibility issues. Ultimately, we opted for the Drag-and-Drop Batch Upload for its significant user experience benefits, accepting the trade-offs related to development complexity and compatibility. This decision paves the way for future enhancements, including preview thumbnails and progress bars, further improving the interface's functionality and user-friendliness**.**
2. **Photo Analysis:** We're tackling the challenge of efficiently extracting crucial parameters for color correction in photos. We considered two main approaches: full pixel analysis, which processes every pixel for maximum detail but is slow and computationally heavy,

and sampling-based analysis, which is quicker but might overlook finer details due to its sparser examination. Our chosen path is adaptive sampling analysis. This method starts with a sampling approach and smartly increases detail inspection in areas with high contrast or variation. It strikes a balance between speed and accuracy, although it introduces more complexity in coding compared to straightforward sampling.

3. **Database Connection Details:** For securely storing database credentials in connectionDetails, various solutions were considered. Storing credentials directly in the application code is the simplest but poses a huge security risk. Using a configuration file improves separation but remains insecure if accessed. Environment variables offer better security, managed outside application files, yet can be compromised with system access. The chosen solution is an encrypted configuration file combined with a key vault. This approach balances security and practicality by protecting credentials at rest through encryption and managing decryption keys via a secure storage solution. However, it introduces more setup complexity and necessitates a key management system.

4. **User Photo Editing Review (reviewEdits):**For the User Photo Editing Review feature, addressing the need for users to compare edited photos against originals led to considering an overlay versus a side-by-side display. The overlay, while space-efficient, posed challenges in discerning subtle changes. Therefore, the side-by-side view was chosen for its clear visualization of edits, accepting the tradeoff of requiring more screen space and potentially necessitating a responsive design for smaller screens. This approach emphasizes clarity and user confidence in approving edits, especially for subtle color corrections.

5. **Evaluation Metrics**: Defining appropriate evaluation metrics for assessing the effectiveness of color auto-correction algorithms was crucial. We considered metrics such as color accuracy, contrast enhancement, and perceptual quality. Ultimately, we adopted a combination of quantitative metrics and qualitative feedback to evaluate the performance of the algorithm.

6. **Color Correction Presets:** We're introducing a Color Correction Presets feature, known as adjustmentPreset, to offer quick color fixes for our users. While fully automated correction provides speed, its one-size-fits-all approach might not hit the mark in every lighting scenario. We considered two main paths: purely Automated Presets, offering rapid adjustments but potentially missing the mark in unique conditions, and User-Editable Presets, which allow for finer control but complicate the user experience and lengthen the editing process. Our chosen path is a hybrid approach: start with Automated Presets for speed, then allow users to tweak those presets for precision. This strikes a balance between convenience and customization, aiming to please both casual users and those with more particular needs. The main tradeoff here is ensuring the user-editing interface is intuitive, to keep the process smooth and user-friendly.

7. **Training Data**: An issue arose regarding the availability and quality of training data for the machine learning model. We explored different sources for collecting a diverse dataset covering various lighting conditions, color temperatures, and image types. We decided to use adobe 5k dataset and dataset collected from photographers.

8. **Algorithm Selection**: One of the initial decisions was to choose the algorithm for color auto-correction. We considered various options such as histogram equalization, contrast stretching, and machine learning-based approaches. We opted for a machine learning-based approach due to its ability to handle complex color transformations and adapt to diverse image characteristics.

9. **Error Handling and Robustness:** Dealing with errors and edge cases during color auto-correction posed a significant design consideration. We implemented error handling mechanisms to handle exceptions, such as invalid input formats, out-of-range pixel values, and memory errors.