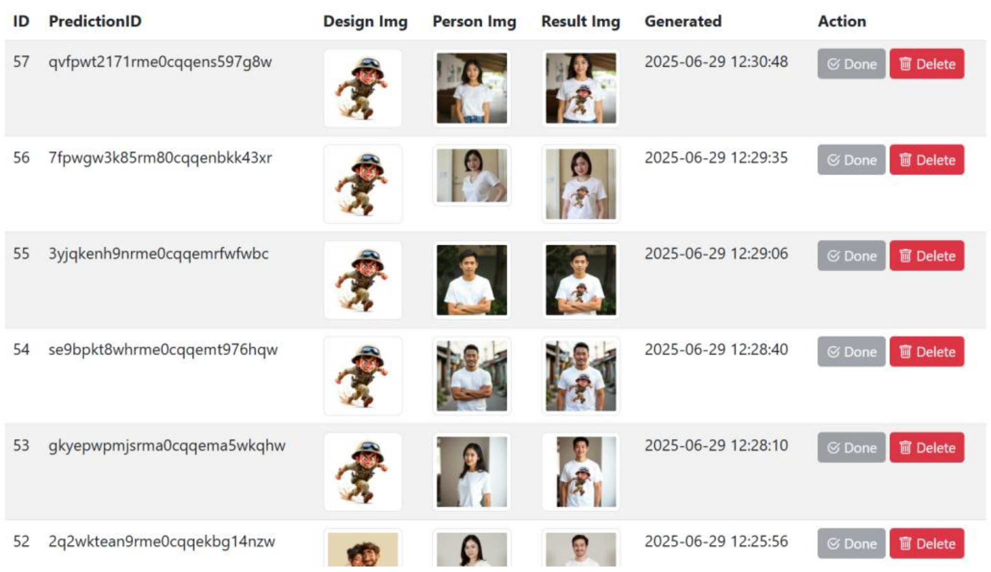
Contents

🧾 AI T-Shirt Design Application – Documentation (EN)

🌟 Introduction

This is a t-shirt design application that allows you to generate a **realistic visual of a person wearing your custom t-shirt design** using AI. It simplifies the process of visualizing and presenting your digital t-shirt artwork.

As a side hustle, you can become a freelance t-shirt designer. Just find a local DTG (Direct to Garment) t-shirt printing vendor near you. Compare prices, shirt colors, materials, and sizes. For each customer order, you can set your own profit margin between 10% to 30%.



ID	PredictionID	Design Img	Person Img	Result Img	Generated	Action
57	qvfpwt2171rme0cqqens597g8w				2025-06-29 12:30:48	Done Delete
56	7fpwgw3k85rm80cqqenbkk43xr				2025-06-29 12:29:35	Done Delete
55	3yjqkenh9nrme0cqqemrfwfwbc				2025-06-29 12:29:06	Done Delete
54	se9bpkt8whrme0cqqemt976hqw				2025-06-29 12:28:40	Done Delete
53	gkyepwpmjsrma0cqqema5wkqhw				2025-06-29 12:28:10	Done Delete
52	2q2wktean9rme0cqqekbg14nzw				2025-06-29 12:25:56	Done Delete

You can upload:

- Your t-shirt design (artwork)
- A person’s photo (endorser or model wearing a plain t-shirt)

The AI will merge your digital design with the model photo, resulting in a **mock-up of a t-shirt** worn by that person.

Ensure your master design file is in high resolution (hi-res) to maintain quality for print or display.

Demo video:

https://www.youtube.com/watch?v=JomXxoUOLRo

🛠 Requirements

To use this application, you must:

- Have your own hosting and domain
- Have a Replicate.com API Token (get it at https://replicate.com/account/api-tokens)
- Understand basic PHP & MySQL and how to host PHP applications

---

🚀 Installation Guide

Follow these steps to set up your **AI T-Shirt Design Application**:

---

1. Upload the Application

- **Linux (Apache server)**
  Upload designkaos.zip to:
  /var/www/html/yourfolder
- **Windows (XAMPP)**
  Upload designkaos.zip to:
  C:/xampp/htdocs/yourfolder

Replace yourfolder with your desired app folder name (e.g. kaosai)

---

2. Create the Database

- Create a new MySQL database (you can name it designkaos or anything else)
- Use **phpMyAdmin** to import the SQL file:
  sql/designkaos.sql

---

3. Edit Database Configuration

- Open config.php located in your root folder
- Set the database connection to match your setup:

```
'db' => [
    'host'     => 'localhost',
    'dbname'   => 'designkaos',   // your database name
    'user'     => 'root',         // your database user
    'password' => '',             // your database password
    'charset'  => 'utf8mb4',
],
```

✅ 4. Edit .htaccess to Match Your Folder Name

- Open the .htaccess file in the root folder

- Update this line:

RewriteBase /yourfolder/

Replace /yourfolder/ with your actual folder name
—for example, if your app is in kaosai:

RewriteBase /kaosai/

RewriteCond %{REQUEST_URI} ^/kaosai/?$ [NC]

RewriteRule ^$ public/login.php [L,R=302]

This ensures URL redirection works correctly in browsers.

---

5. Create Admin Account

- Visit the setup script in your browser:
  http://localhost/yourfolder/public/create_admin.php
- Set your initial admin username and password.

---

6. Login to Admin Dashboard

- Visit the login page:
  https://yourdomain.com/yourfolder/public/login.php

---

7. Set API Key & Host

- After login, go to **Settings** menu
- Enter your:
  - **Replicate API Token**
  - **Application base URL** (e.g. https://yourdomain.com/yourfolder/)

---

8. Delete Logs (Optional)

- Use the **Delete Logs** menu in the dashboard to clear old log files.

⚠️ Important Tips

🔒 Admin Dashboard is Private!

The admin dashboard is only for you as the app owner.
Do NOT share the login access with anyone else.

Admin access allows full control of the design, image generation, file deletion, and API usage.

💵 Replicate API Cost

Using Replicate API to generate images is **not free**.

Each image costs around **$0.02 to $0.06 USD** depending on the model and resolution.

Make sure to:

- Monitor usage regularly.
- Set a budget or usage limit.
- Avoid unnecessary image generations during testing.

📦 Database Structure

🗄 Database: designkaos

This database is used by your **AI T-Shirt Design Application** to store:

- Uploaded T-shirt designs
- Admin login credentials
- Person (model) images
- Generated AI image results
- Application settings (API keys, domain, etc.)

📁 1. Table: masterdesign

Stores uploaded **T-shirt design data**.

Column	Type	Description
id	int, AUTO_INCREMENT	Unique ID for each design

Column	Type	Description
titledesign	varchar(255)	Title or name of the design
imagesdesign	varchar(255)	File name of the uploaded design image
ispublish	tinyint(1)	1 = published, 0 = hidden
submitdate	Datetime	When the design was uploaded

🔑 **Primary Key**: id

👤 2. Table: msadmin

Stores **admin login credentials**.

Column	Type	Description
adminid	int, AUTO_INCREMENT	Unique ID for admin
loginadmin	varchar(255)	Admin username
loginpassword	Text	Hashed password (never plaintext)

🔑 **Primary Key**: adminid

🧍 3. Table: person

Stores **person/model images** used for T-shirt mockups.

Column	Type	Description
id	int, AUTO_INCREMENT	Unique ID for each person
titleperson	varchar(255)	Name or label for the person
person_images	Text	File name of the uploaded person image
submitdate	Datetime	Upload timestamp

🔑 **Primary Key**: id

🧪 4. Table: photosample

Stores **AI-generated image combinations** between a design and a person.

Column	Type	Description
id	int, AUTO_INCREMENT	Unique ID
predictionid	Text	ID returned by the Replicate API
designid	Int	Foreign key referring to masterdesign.id
personid	Int	Foreign key referring to person.id
urlimagesdesign	Text	URL/path of the design image
urlimagesperson	Text	URL/path of the person image
imagesresult	Text	URL/path of the AI-generated result
ispublish	tinyint(1)	1 = published, 0 = private
generateddate	Datetime	When the image was generated

🔑 **Primary Key**: id

⚙️ 5. Table: settings

Stores **application configuration settings**, like API tokens and domain names.

Column	Type	Description
id	int, AUTO_INCREMENT	Unique ID
key	varchar(255)	Configuration key name (e.g., API token)
value	Text	Configuration value (e.g., domain URL)

🔑 **Primary Key**: id

📝 **Example entries:**

(1, 'REPLICATE_API_TOKEN', 'your_token_here'),

(2, 'HOST_DOMAIN', 'https://yourdomain.com/yourfolder/public');

🔑 Summary of Primary Keys and Auto-Increment

Table	Primary Key	Auto-Increment	Description
masterdesign	id	Yes (next: 28)	T-shirt design entries
msadmin	adminid	Yes (next: 3)	Admin account info
person	id	Yes (next: 16)	Person/model image data
photosample	id	Yes (next: 79)	AI-generated design + model results
settings	id	Yes (next: 3)	App-wide config (tokens, URLs, etc.)

✅ **Additional Notes:**

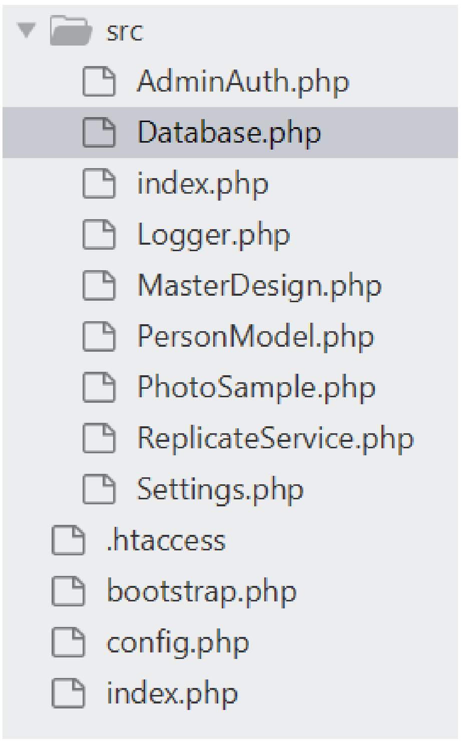
- All tables use **InnoDB** engine and utf8mb4 charset for full Unicode support.
- Each entity (design, person, sample) is stored with timestamps for sorting.
- The photosample table acts as a **junction table** linking designs and persons, plus AI output.
- The settings table is flexible, allowing easy updates to tokens, domains, and environment configs.

📁 Project Structure

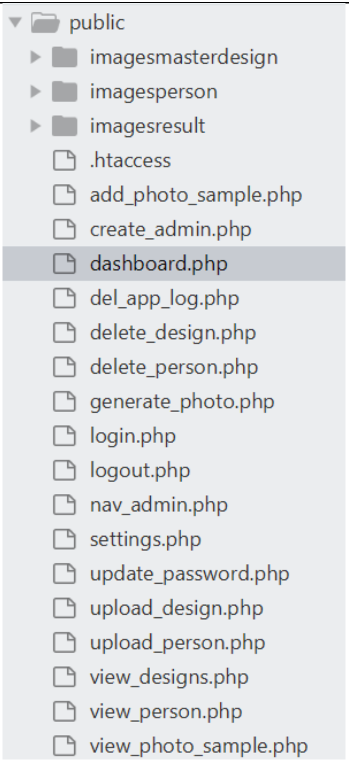
```
/your-app-folder/
|
├── public/        ← user-accessible PHP files
|   ├── login.php, dashboard.php, upload_design.php, etc.
|
├── src/          ← backend classes & logic
|   ├── Database.php, Logger.php, AdminAuth.php, etc.
|
├── sql/          ← SQL file for DB setup
├── logs/          ← Application logs
├── documentation/ ← User docs (optional)
```

├── .htaccess ← Redirects

├── index.php ← Redirect to login/dashboard



```
▼ src
    AdminAuth.php
    Database.php
    index.php
    Logger.php
    MasterDesign.php
    PersonModel.php
    PhotoSample.php
    ReplicateService.php
    Settings.php
  .htaccess
  bootstrap.php
  config.php
  index.php
```



```
▼ public
  ▶ imagesmasterdesign
  ▶ imagesperson
  ▶ imagesresult
    .htaccess
    add_photo_sample.php
    create_admin.php
    dashboard.php
    del_app_log.php
    delete_design.php
    delete_person.php
    generate_photo.php
    login.php
    logout.php
    nav_admin.php
    settings.php
    update_password.php
    upload_design.php
    upload_person.php
    view_designs.php
    view_person.php
    view_photo_sample.php
```

📁 **Important Folders inside /public**

- imagesmasterdesign – for your digital t-shirt design artwork (.jpg, .png, .webp)
- imagesperson – for photos of endorsers wearing blank t-shirts
- imagesresult – AI-generated output combining the design and endorser

---

📜 **Main Files Explained**

2.2.1. public/add_photo_sample.php

Admin form to select:

- A master design
- An endorser photo

It stores this combination into the photosample table and redirects to view_photo_sample.php. These are later processed by the AI into a final visual mock-up.

2.2.2. public/create_admin.php

Web-based form to:

- Create a new admin account
- Update existing admin password

This file is typically used during the first install. Once configured, you should either delete it or disable it in config.php:

```
'features' => [
    'admin_password' => false,  // true = allow setup, false = disable setup
],
```

2.2.3. public/dashboard.php

Admin dashboard displaying:

- App overview (ID + EN)
- Features & benefits
- Installation instructions
- Contact links (WhatsApp, Instagram, LinkedIn)

2.2.4. public/del_app_log.php

Page for admins to manually delete logs/app.log.
Useful for cleaning old logs and managing disk space.

2.2.5. public/delete_design.php

Deletes a master design based on the id in the URL.
Also removes the design image file and logs the action.

Security features:

- Requires admin login
- Integer-only ID sanitization
- Logs deletion activity

2.2.6. public/delete_person.php

Deletes a person/endorser record.
Also removes the associated photo file and logs the activity.

🧠 **Tables Involved**

- masterdesign – stores uploaded t-shirt designs
- person – stores uploaded endorser photos
- photosample – stores design + person combinations
- msadmin – stores admin credentials

2.2.7. public/generate_photo.php

📄 **File Description**

The file public/generate_photo.php serves as the **main backend script** responsible for generating a mock-up image by combining a t-shirt design and a person's photo (endorser), using **AI from Replicate.com**. The output is then automatically downloaded and stored on the server.

🎯 **Primary Objectives**

- Fetch sample data (photosample) by ID
- Send a request to the Replicate AI API with both the design and person photo
- Download the AI-generated result image
- Save the resulting image in the imagesresult/ folder
- Save the result path in the photosample database table
- Redirect back to the view_photo_sample.php page

🔍 **Step-by-Step Explanation**

✅ **1. Admin Access Protection**

```
session_start();
if (empty($_SESSION['admin'])) {
  header('Location: login.php'); exit;
}
```

Only authenticated admin users are allowed to execute this script.

✅ 2. Initialization

```
$settings = new Settings($db);
$token    = $settings->get('REPLICATE_API_TOKEN');
$ps       = new PhotoSample($db);
$rs       = new ReplicateService($token);
```

Initializes settings and loads the API token from the database to prepare for communication with Replicate.com.

✅ 3. Validate Input and Fetch Sample Record

```
$id = (int)($_POST['id'] ?? 0);
$sample = $db->fetch('SELECT * FROM photosample WHERE id = :id', [':id'=>$id]);
```

Ensures a valid sample ID is provided and retrieves the corresponding record from the database.

✅ 4. Prepare Image URLs (with Public Host Domain)

```
$host = rtrim($settings->get('HOST_DOMAIN'), '/');
$sample['urlimagesdesign'] = $host . '/' . ltrim($sample['urlimagesdesign'], '/');
$sample['urlimagesperson'] = $host . '/' . ltrim($sample['urlimagesperson'], '/');
```

These URLs ensure that the images are accessible publicly for Replicate.com’s API to fetch.

✅ 5. Send Prompt to Replicate API

```
$prompt = "Put the person into a plain t-shirt with the medium design on it. Size of design is medium.";
$prediction = $rs->generateAndWait($sample['urlimagesdesign'], $sample['urlimagesperson'], $prompt);
```

The API is instructed to combine the two images with a defined prompt to produce a realistic mockup.

✅ 6. Save Prediction ID

```
$ps->updatePredictionId($id, $prediction['id']);
```

The prediction ID returned by Replicate is saved for logging or future tracking.

✅ 7. Download AI Output Image

$outputUrl = $prediction['output'] ?? null;

$data = file_get_contents($outputUrl);

Fetches the final mock-up image from Replicate's CDN.

✅ 8. Save the Image to Local Folder

$fname = $id . '_' . bin2hex(random_bytes(4)) . '.' . $ext;

file_put_contents($local, $data);

The image is stored locally in imagesresult/, using a randomized filename for uniqueness.

✅ 9. Update Result Path to Database

$ps->updateResult($id, 'imagesresult/' . $fname);

Stores the relative path of the output image in the database.

✅ 10. Redirect to Result Page

header('Location: view_photo_sample.php');

User is redirected to view the list of AI-generated mockups.

📁 Involved Folders

Folder	Purpose
imagesmasterdesign/	Stores digital t-shirt design images
imagesperson/	Stores photos of models/endorsers
imagesresult/	Stores AI-generated output images

🛡️ Security & Stability

Feature	Status
✅ Admin authentication	Enforced
✅ Numeric ID validation	Safe
✅ Exception handling/log	Included
✅ Auto folder creation	Supported
✅ Unique file naming	Secured

✅ Conclusion

The generate_photo.php file acts as the **core AI pipeline** in this application.
It handles the **entire process from data fetching, prompt generation, AI rendering, image downloading, to database updating and user redirection**.

Its well-structured design makes it highly reliable, secure, and essential for any AI-powered mock-up system.

2.2.8. public/login.php

📄 File Description

The file public/login.php provides a clean, secure, and responsive **admin login interface**. This form validates user credentials against the msadmin database using AdminAuth::verify() and starts a session upon success.

🎯 Primary Objectives

- Allow admin to log in securely
- Start a session for authenticated access
- Redirect successful logins to the dashboard (dashboard.php)
- Display error message on failure

🔍 Step-by-Step Explanation

✅ 1. Session Start & Form Handling

session_start();

Initiates a session. If the form is submitted using POST, the script retrieves and trims the username and password inputs.

✅ 2. Verification

```
if ($adminAuth->verify($u, $p)) {
  $_SESSION['admin'] = $u;
  $_SESSION['login_time'] = date('Y-m-d H:i:s');
  header('Location: dashboard.php');
  exit;
}
```

- Credentials are verified using AdminAuth::verify()
- On success:
    - Session variable admin is set
    - Current timestamp is stored in login_time
    - Redirects to the admin dashboard

❌ If Verification Fails

$err = 'Login gagal. Username atau password salah.';

Logger::warning("Login failed for {$u}");

An error message is shown, and a log entry is written.

💻 HTML UI Features

- Built using **Bootstrap 5**
- Includes:
    - Username and Password input fields
    - Password visibility toggle button (👁 icon)
    - Error alert on failed login
- Styled for responsiveness

🛡️ Security Features

Feature	Status
✅ Prepared input	Yes (via verify logic)
✅ Server-side validation	Yes
✅ Password hash checking	Yes (password_verify)
✅ Session protection	Yes
✅ Error logging	Yes

🖱️ UI Features

Component	Description
Username input	Required field for login name
Password input	Required field, hidden by default
Eye icon toggle	Reveals password for user convenience
Submit button	Triggers authentication logic
Error alert	Shows feedback on incorrect login attempt

✅ Conclusion

The login.php file provides a **professional and secure** admin login mechanism with a responsive UI. It is a critical part of any backend system and is built to integrate tightly with the rest of the admin panel.

2.2.9. public/logout.php

📄 File Description

The file public/logout.php is a **simple and essential logout script** for ending the current admin session. It ensures that after logging out, the user is redirected back to the login page and loses access to all authenticated admin features.

🎯 Primary Objectives

- Terminate the current session
- Log the user out securely
- Redirect the user to login.php

🔍 Code Breakdown

✅ 1. Start the Session

```
session_start();
```

Initializes the session, required in order to destroy it.

✅ 2. Destroy the Session

```
session_destroy();
```

Clears all session data, effectively logging the user out.

✅ 3. Redirect to Login Page

```
header('Location: login.php');

exit;
```

Redirects the user to the login page after logout is complete.

🛡️ Security Benefits

Feature	Status
✅ Ends admin session	Yes
✅ Prevents back-button access	Yes (after logout, protected pages redirect to login)
✅ Clean and fast	Yes

✅ Conclusion

The logout.php script provides a **clean and effective** mechanism for terminating an admin session. It is essential in maintaining session-based access control and contributes to the overall security of the application.

Its simplicity ensures:

- Fast logout
- No residual session data
- Smooth redirect experience

2.2.10. public/nav_admin.php

📄 File Description

The file nav_admin.php provides the **main admin sidebar navigation** for the application. It is included on most admin pages to ensure consistent navigation between sections.

This sidebar uses **Bootstrap 5** components and **Bootstrap Icons** to offer a clean, collapsible, and user-friendly menu layout.

🧩 Function and Role

- Acts as the **primary navigation UI** for the admin panel
- Allows quick access to all major features, including:
    - Dashboard
    - Settings
    - Design and Person upload & view
    - Photo mockup generation
    - Password updates and logout
- Structured as a **Bootstrap accordion menu** with icons for intuitive access

---

🔍 Menu Structure Overview

Menu Item	Icon	Description
Dashboard	bi-speedometer2	Redirects to the admin home screen

Menu Item	Icon	Description
Settings	bi-sliders	Manage API tokens and domain configuration
Upload Design	bi-file-earmark-arrow-up	Upload new t-shirt artwork
View Designs	bi-palette	View existing uploaded designs
Upload Person	bi-person-plus-fill	Upload new model/endorser photos
View Person	bi-people	Manage and review uploaded person images
Add Photo Sample	bi-image	Combine a design + person into a photo sample
View Photo Sample	bi-images	See list of combined AI mockups
Delete Logs	bi-trash3-fill	Remove app log file to free up space
Update Password	bi-shield-lock-fill	Change admin account password
Logout	bi-box-arrow-right	Safely sign out of the admin panel

🖼️ UI Behavior

- Uses <div class="collapse show" id="menuAdmin"> to allow for collapsible menu expansion
- <i class="bi ..."> adds intuitive visual icons
- The active section is indented using Bootstrap's ps-5 (padding start) for clarity

---

🛡️ Security Note

While this file only controls UI navigation, **actual page access security is handled individually** in each PHP file using session_start() and $_SESSION['admin'] checks.

✅ Conclusion

The nav_admin.php file is a vital component that ensures the **admin interface is easy to navigate**, well-organized, and professional-looking. Its collapsible sidebar structure paired with iconography creates an excellent user experience.

This modular layout also makes it easy to extend—additional features can be added with just one <a> line.

2.2.11. public/settings.php

📄 **File Description**

The settings.php file allows the administrator to **configure and update application-level settings**, including:

- The **Replicate API token** used to generate AI-based mockups
- The **Host Domain URL** to ensure image paths are publicly accessible

---

🧩 **Function and Role**

- Secured behind an admin session ($_SESSION['admin'])
- Renders a settings form where the administrator can:
  - View and update the Replicate API token
  - Update the public host domain
- Provides form validation and success/error messaging
- Logs actions and database changes via the built-in Logger

🧪 **Main Features**

Feature	**Description**
Admin Auth Required	Redirects unauthorized users to the login page
Form Handling	Accepts and processes POST submissions from the form
Token Visibility Toggle	Includes an eye icon to show/hide the API token
Validation & Error Log	Detects empty input, handles PDO exceptions, and logs issues
Database Persistence	Stores settings using $settings->set(key, value) method
Current Settings Loaded	Retrieves and displays current saved values on page load

---

🖼️ User Interface (UI) Elements

- **Input: Replicate API Token**
  - Rendered as a password field for security
  - Toggleable with an eye icon (Bootstrap icons)
- **Input: Host Domain**
  - Used to prepend URLs so Replicate API can access images
- **Save Button**
  - Submits the form using POST method

🛡️ Security Considerations

Area	Status	Notes
Admin authentication	✅ Required	Only accessible after login
Input sanitization	✅ Yes	All fields trimmed and validated
XSS prevention	✅ Yes	All user inputs/output passed through htmlspecialchars()
Logging	✅ Active	Logs success and error messages
Password visibility toggle	✅ Optional	Controlled via JS toggle

---

🧾 Sample Logging Events

- Logger::info('Settings updated: REPLICATE_API_TOKEN or HOST_DOMAIN.')
- Logger::error('Settings: DB error while updating settings: ...')

These help track system changes and diagnose issues.

🧩 Related Folders and Files

- **Settings Table / Class**: Stores key-value settings for token & domain
- **nav_admin.php**: Included sidebar navigation for consistent UI

- **bootstrap.php**: Handles DB connection and autoloaders

✅ Conclusion

The settings.php file provides a **secure, intuitive configuration panel** for administrators to manage essential application settings related to AI functionality and public image hosting. Its minimalistic layout, combined with error handling and session control, ensures both usability and safety.

2.2.12. public/update_password.php

📄 File Description

The update_password.php file enables the administrator to securely update their account password through a protected form interface.

🧩 Function and Role

- Checks for authenticated sessions via $_SESSION['admin']
- Displays the current admin username (read-only)
- Validates the old password before accepting a new one
- Provides real-time feedback with success and error messages
- Offers a toggle feature for showing/hiding passwords

🔐 Key Features

Feature	Description
Admin Login Required	Ensures access is limited to authenticated users
Password Verification	Verifies old password before accepting a new one using $adminAuth->verify()
Secure Update Method	Uses $adminAuth->createOrUpdatePassword() to save the new password

Feature	Description
Toggle Visibility Icons	Built-in Bootstrap Icons to toggle password input visibility
Feedback Mechanism	Alerts user to success or failure of update

---

📝 Form Fields

- **Username** (non-editable): Pulled from session variable $_SESSION['admin']
- **Old Password**: Required for verification
- **New Password**: Required, must not be empty

💬 Validation Rules

Field	Rule
oldpass	Must match existing password
newpass	Must not be empty

🛡️ Security Considerations

Aspect	Status	Notes
Session Authentication	✅ Yes	Redirects to login.php if not logged in
Input Validation	✅ Yes	Validates old password and ensures new is not empty
HTML Escaping	✅ Yes	Uses htmlspecialchars() on all output
JS Password Toggle	✅ Yes	Uses Bootstrap icons to toggle password visibility

---

🧩 Related Components

Component	Role
adminAuth class	Handles password verification & updating

Component	Role
nav_admin.php	Injected sidebar navigation
bootstrap.php	Provides DB connection and class autoloaders

✅ Conclusion

The update_password.php file provides a **secure and user-friendly interface** for administrators to update their login credentials. It implements essential checks to ensure old password correctness and secure storage of the new password using the system's adminAuth utility.

2.2.13. public/upload_design.php

📄 File Description

The upload_design.php script provides an administrative interface to upload new **master t-shirt designs** into the system. It handles file validation, naming, storage, and database insertion.

🧩 Function and Role

- Accessible **only to logged-in admins**.
- Accepts **design title** and **image file** as input.
- Validates file extension (only JPG, JPEG, PNG, WEBP).
- Renames the uploaded file using the pattern: [next_id]_[RANDOM8].[ext]
- Inserts the new design into the database.

📤 Form Fields

Field Name	Type	Required	Notes
titledesign	text	✅ Yes	Title of the design
imagesdesign	file	✅ Yes	Allowed formats: JPG, JPEG, PNG, WEBP

🧾 Validation Rules

Check	Description
Title must not be empty	Form will return error if left blank
File must be uploaded	Upload required
Extension must be valid	Only jpg, jpeg, png, webp allowed
File must be successfully moved	Uses move_uploaded_file() to copy file into imagesmasterdesign/
DB insert must succeed	Calls $md->insert() after successful file move

---

📦 File Naming Format

- **Generated file name:** [AUTO_INCREMENT_ID]_[RANDOM8].[EXT]
- Example: 13_XYZPQRST.webp

This reduces file name collision and helps organize uploads.

✅ Feedback Mechanism

Status	Message
✅	**"Design uploaded successfully."**
❌	**"Design title is required."**
❌	**"Image file must be uploaded."**
❌	**"Only JPG, JPEG, PNG, or WEBP formats allowed."**
❌	**"Failed to move uploaded file."**
❌	**"Failed to save data to the database."**

🧩 Related Components

Component	Purpose
MasterDesign	Handles the database insert logic
bootstrap.php	Loads DB connection and app settings
nav_admin.php	Loads the sidebar navigation

🔐 Security Considerations

Concern	Mitigated?	Details
Admin authentication required	✅ Yes	Session $_SESSION['admin'] checked
File upload validation	✅ Yes	MIME and extension checking, unique name generation
Output escaping	✅ Yes	Uses htmlspecialchars() to sanitize user input

---

📷 Screenshot Suggestion (for report)

If including screenshots in a Word document:

- Show the upload form interface (Title Design + Upload Image)
- Display example of success and error message
- Showcase file renamed format

✅ Conclusion

The upload_design.php script ensures that only valid and titled design files are stored in a secure, structured, and trackable manner. It enhances the admin's capability to manage the collection of design assets efficiently and safely.

2.2.14. public/upload_person.php

Its function is to **upload a photo of a model/endorser** into the system. It accepts input in the form of:

- The name or title of the person (titleperson)
- An image file of the person (person_images)

Then it saves the image in the /imagesperson folder and records the metadata into the database using the PersonModel.

🔧 Section by Section Breakdown

1. Header, Session & File Setup

```
require __DIR__ . '/../bootstrap.php';
session_start();
if (empty($_SESSION['admin'])) {
   header('Location: login.php');
   exit;
}
```

- bootstrap.php: Sets up the database connection and autoload.
- session_start(): Starts a session for admin login validation.
- If the user is not logged in as admin, they are redirected to the login page.

---

2. Initialize Model and Variables

```
$pm = new PersonModel($db, __DIR__ . '/imagesperson');
$errors  = [];
$success = null;
```

- Creates a $pm object of class PersonModel to handle database insertions.
- Initializes variables to store any error or success messages.

---

3. Process Form Submission

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
```

- Executes this block only when the form is submitted via POST.

a. Input Validation

```
$title = trim($_POST['titleperson'] ?? '');
```

```
if ($title === '') {
    $errors[] = 'Title person is required.';
}
if (empty($_FILES['person_images']['name'])) {
    $errors[] = 'Image file must be uploaded.';
}
```

- Ensures the title is not empty.
- Ensures a file was uploaded.

b. Image Format Validation

```
$ext = strtolower(pathinfo($_FILES['person_images']['name'], PATHINFO_EXTENSION));
if (!in_array($ext, ['jpg','jpeg','png'])) {
    $errors[] = 'Only JPG, JPEG, or PNG formats are allowed.';
}
```

- Checks that the file extension is one of the allowed image types.

c. Generate New File Name

```
$row = $db->fetch('SELECT COALESCE(MAX(id),0)+1 AS nextid FROM person');
$id  = $row['nextid'];
$rand = substr(str_shuffle('ABCDEFGHIJKLMNOPQRSTUVWXYZ'), 0, 8);
$newName = $id . '_' . $rand . '.' . $ext;
```

- Gets the next ID for the person from the database.
- Creates a unique file name using the ID and a random 8-letter uppercase string.

d. Move File & Save to DB

```
$dest = __DIR__ . '/imagesperson/' . $newName;
if (move_uploaded_file($_FILES['person_images']['tmp_name'], $dest)) {
    if ($pm->insert($title, $newName)) {
        $success = 'Person image successfully uploaded.';
    } else {
        $errors[] = 'Failed to save data to the database.';
        unlink($dest);
    }
} else {
    $errors[] = 'Failed to move uploaded file.';
}
```

- move_uploaded_file moves the uploaded file to the target directory.
- If successful, the image metadata is inserted into the database.
- If the insert fails, the uploaded file is deleted.

🧾 HTML Section

Displays the upload form and the result.

a. Error and Success Messages

```
<?php if ($errors): ?>
  <div class="alert alert-danger"><ul>...<li><?= htmlspecialchars($e) ?></li>...</ul></div>
<?php endif; ?>
<?php if ($success): ?>
  <div class="alert alert-success"><?= htmlspecialchars($success) ?></div>
<?php endif; ?>
```

- Shows error messages in a red alert box.
- Shows a success message in a green alert box if the upload is successful.

b. Form Input

```
<form method="post" enctype="multipart/form-data">
  <input type="text" name="titleperson">
  <input type="file" name="person_images">
  <button type="submit">Upload</button>
</form>
```

- enctype="multipart/form-data" is required for file uploads.
- Allows admin to enter a title and choose an image file to upload.

🎯 Summary of Functionality

The upload_person.php file:

1. Provides a form to upload a person endorser (name + image).
2. Validates the input and file format.
3. Moves the uploaded file to /imagesperson/.
4. Inserts metadata (title & file name) into the database.
5. Displays success or error messages accordingly.

2.2.15. public/view_designs.php

📃 Main Purpose

This page is used to **display a list of T-shirt designs (master designs)** from the database in a table format, specifically for admins. Each row shows:

- Design ID
- Design title
- Thumbnail image
- Publish status (Yes/No)
- Submit date
- Action button to delete the design

🧱 Structure and Code Explanation

1. Header and Authentication

```
require __DIR__ . '/../bootstrap.php';
session_start();
if (empty($_SESSION['admin'])) {
   header('Location: login.php');
   exit;
}
```

- Loads basic configuration from bootstrap.php.
- Checks whether an admin is logged in. If not, redirects to the login.php page.

---

2. Fetching Data from the Database

```
$md = new MasterDesign($db, __DIR__ . '/imagesmasterdesign');

$designs = $md->fetchAll();
```

- Creates an instance of the MasterDesign class to handle design data.
- Retrieves all design records from the database using fetchAll().

---

3. HTML Table Display

```
<h1 class="mb-4">Master Designs</h1>

<table class="table table-striped table-hover">
```

- Displays a page title and a styled Bootstrap table.
- The columns shown are:
  - ID
  - Title
  - Thumbnail image
  - Publish status
  - Submission date
  - Action (delete)

```
<?php foreach ($designs as $d): ?>

<tr>

  <td><?= $d['id'] ?></td>

  <td><?= htmlspecialchars($d['titledesign']) ?></td>

  ...
```

- Loops through each design and renders a row in the table.
- Displays a small (80px wide) thumbnail image. When clicked, it opens a **modal popup** to preview the full image.

---

4. Delete Action Button

```
<a href="delete_design.php?id=<?= $d['id'] ?>" class="btn btn-sm btn-danger" onclick="return confirm('Are you sure you want to delete this design?');">

  <i class="bi bi-trash"></i> Delete

</a>
```

- Provides a button to delete the corresponding design.
- Uses confirm() JavaScript function to show a confirmation dialog before proceeding.

---

5. Modal for Image Preview

```
<div class="modal fade" id="designModal" ...>
```

- When a thumbnail is clicked, this modal appears showing the full-size image.
- Uses Bootstrap modal components and data-bs-* attributes.

---

6. JavaScript for Modal Logic

```
var designModal = document.getElementById('designModal');
designModal.addEventListener('show.bs.modal', function (event) {
  var trigger = event.relatedTarget;
  var src = trigger.getAttribute('data-src');
  var modalImg = designModal.querySelector('#modalDesignImage');
  modalImg.src = src;
});
```

- Listens for the event when the modal is triggered.
- Dynamically sets the image source in the modal based on the data-src attribute from the thumbnail.

🧠 Summary

view_designs.php is an admin dashboard page that:

- Displays a list of T-shirt designs from the database.
- Allows previewing full-size images in a modal.
- Provides a delete option for each design.
- Uses Bootstrap and Bootstrap Icons for a modern interface.

2.2.16. public/view_person.php

📃 Main Purpose

This script provides an **admin page** that displays a list of **person endorsers** (people who model the T-shirt designs) in a table format. Each entry includes:

- ID
- Title/Name of the person
- Image thumbnail
- Submission date
- A delete button

It also allows the admin to preview a full-size image of the person in a modal (popup window).

🔍 Code Breakdown

1. Authentication and Initialization

```
require __DIR__ . '/../bootstrap.php';

session_start();

if (empty($_SESSION['admin'])) {

   header('Location: login.php');

   exit;

}
```

- Includes the core configuration file (bootstrap.php).
- Starts a session and checks if the user is an admin.
- If the admin is not logged in, it redirects them to the login page.

---

2. Data Fetching

```
$pm = new PersonModel($db, __DIR__ . '/imagesperson');

$persons = $pm->fetchAll();
```

- Creates an instance of the PersonModel class to handle database operations for persons.
- Calls fetchAll() to get all person records from the database.

---

3. HTML Output

```
<!DOCTYPE html>

<html lang="id">

<head>...</head>
```

- Standard HTML5 page setup with Bootstrap 5 for layout and styling.
- Language is set to Indonesian (lang="id"), but you can change it to "en" if needed.

---

4. Table of Persons

```
<h1 class="mb-4">Person Endorsers</h1>

<table class="table table-striped table-hover">...</table>
```

- Displays a title and a Bootstrap-styled table.
- Table columns:
  - **ID:** Unique identifier for the person.
  - **Title Person:** The name or title entered for the person.
  - **Image:** A thumbnail of the person’s photo.
  - **Submit Date:** When the person data was submitted.
  - **Action (Aksi):** A delete button.

5. Looping Through Data

```
<?php foreach ($persons as $p): ?>

<tr>

  <td><?= $p['id'] ?></td>

  <td><?= htmlspecialchars($p['titleperson']) ?></td>

  ...

</tr>

<?php endforeach; ?>
```

- Loops through each person and prints their information in a table row.
- Uses htmlspecialchars() to avoid XSS vulnerabilities when displaying data.

---

6. Thumbnail with Modal Preview

```
<a href="#" data-bs-toggle="modal" data-bs-target="#imageModal" data-src="imagesperson/<?=
htmlspecialchars($p['person_images']) ?>">
```

<img src="imagesperson/<?= htmlspecialchars($p['person_images']) ?>" width="80" class="img-thumbnail" alt="">

</a>

- Displays the thumbnail image of the person.
- When clicked, it triggers a Bootstrap modal to preview the full-sized image.

---

7. Delete Button

<a href="delete_person.php?id=<?= $p['id'] ?>" class="btn btn-sm btn-danger" onclick="return confirm('Yakin akan menghapus person ini?');">

  <i class="bi bi-trash"></i> Hapus

</a>

- A delete button that links to delete_person.php with the person's ID as a query parameter.
- JavaScript confirm() prompts the user before deletion.

---

8. Image Preview Modal

<div class="modal fade" id="imageModal" ...>

<img src="" id="modalImage" class="img-fluid" alt="Full View">

</div>

- A Bootstrap modal that will display the full-sized image when triggered.

---

9. JavaScript for Modal

var imageModal = document.getElementById('imageModal');

imageModal.addEventListener('show.bs.modal', function (event) {

var trigger = event.relatedTarget;

var src = trigger.getAttribute('data-src');

var modalImg = imageModal.querySelector('#modalImage');

modalImg.src = src;

});

- When a thumbnail is clicked, this script sets the src of the modal image to the full-size version using data-src.

✅ Summary

view_person.php is an admin-only page that:

- Lists all person endorsers (used as T-shirt mockup models).
- Provides image previews in a modal.
- Allows deletion of any person entry.
- Uses **Bootstrap 5** for UI components and **PHP** for data handling and server-side logic.

Perfect for managing model photo entries in an AI-powered T-shirt design application.

2.2.17. public/view_photo_sample.php

🧾 Main Purpose

This PHP page is for the **admin dashboard** in an AI T-shirt design application. It displays a **paginated list of generated photo samples**, showing:

- The original design image,
- The person/model image,
- The result image (AI-generated merge),
- With options to preview, generate (if not done yet), or delete a record.

🔍 Code Breakdown

1. Authentication & Initialization

```
require __DIR__ . '/../bootstrap.php';
session_start();
if (empty($_SESSION['admin'])) {
    header('Location: login.php');
    exit;
}
```

- Includes app setup from bootstrap.php.
- Starts a session and verifies if the user is an admin.
- If not, redirects to the login page.

---

2. Delete Handler

```
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['delete_id'])) {
  ...
}
```

- If the page receives a POST request with a delete_id, it deletes the corresponding photo sample entry from the database.
- After deletion, it sets a success or error message in the session and reloads the current page.

---

3. Pagination Setup

```
$page = isset($_GET['page']) ? max(1, (int)$_GET['page']) : 1;

$perPage = 10;

$offset = ($page - 1) * $perPage;
```

- *Sets up pagination logic: how many entries per page and which page to show.*

```
$total = $db->fetch('SELECT COUNT(*) AS cnt FROM photosample', [])['cnt'];

$pages = ceil($total / $perPage);

$samples = $db->fetchAll(...);
```

- Retrieves total sample count and fetches a subset of results (LIMIT ... OFFSET ...) for current page.

---

4. Table Display

Each row of the table displays:

- **ID**
- **Prediction ID** (related to AI generation)
- **Design Image**
- **Person Image**
- **Result Image**
- **Generated Date**
- **Action Buttons**

foreach ($samples as $s):

- Iterates through each sample and displays a row in the table.
- For each image (design, person, result), if available, a thumbnail is shown that can be clicked to preview in a modal.

---

5. Action Buttons

✅ Generate Button

```
<form method="post" action="generate_photo.php">
  <input type="hidden" name="id" value="<?= $s['id'] ?>">
  <button class="btn btn-sm btn-success">Generate</button>
</form>
```

- If the result hasn't been generated yet (imagesresult is missing), a green "Generate" button is shown.
- Clicking it will submit the sample's ID to generate_photo.php for AI generation.

🗑️ Delete Button

```
<form method="post" action="">
  <input type="hidden" name="delete_id" value="<?= $s['id'] ?>">
  <button class="btn btn-sm btn-danger">Delete</button>
</form>
```

- Deletes the sample after user confirms.
- Uses POST and a confirmation dialog.

---

6. Pagination UI

```
<ul class="pagination">
  <li class="page-item ..."><a class="page-link" href="?page=...">...</a></li>
</ul>
```

- Bootstrap pagination to navigate between pages.

---

7. Image Modal (Preview)

```
<div class="modal fade" id="photoModal">
```

```
  <img id="modalPhotoImage" class="img-fluid">
</div>
```

- A Bootstrap modal that displays a full-size image when a thumbnail is clicked.

---

8. JavaScript Behavior

a. Modal Preview Logic

```
photoModal.addEventListener('show.bs.modal', function (e) {
  var src = e.relatedTarget.getAttribute('data-src');
  photoModal.querySelector('#modalPhotoImage').src = src;
});
```

- When an image thumbnail is clicked, sets the modal image to the appropriate URL.

b. Loading Feedback for Generate Button

```
form.addEventListener('submit', function(e) {
  const btn = form.querySelector('button[type="submit"], button.btn-success');
  if (btn) {
    btn.disabled = true;
    btn.innerHTML = '<i class="bi bi-hourglass-split"></i> Generating...';
  }
});
```

- When the "Generate" button is clicked, it disables the button and changes the label to show a loading spinner.

✅ Summary

view_photo_sample.php is a backend admin page that:

- Shows photo samples of T-shirt designs generated by AI.
- Allows admins to **view, generate**, or **delete** each sample.
- Features **pagination**, **image preview modal**, and **confirmation dialogs**.
- Built using **Bootstrap 5** and **PHP** with a clean and responsive interface.

Ideal for managing AI-generated design previews in a professional T-shirt mockup application.

📂 **src/ Folder Overview**

This folder contains **9 PHP source files** that handle backend logic for an AI-based T-shirt design web application:

```
src/
├── AdminAuth.php
├── Database.php
├── index.php
├── Logger.php
├── MasterDesign.php
├── PersonModel.php
├── PhotoSample.php
├── ReplicateService.php
└── Settings.php
```

2.3.1. src/AdminAuth.php

This class handles **admin authentication functions**, including:

- Creating or updating admin accounts
- Changing admin passwords
- Verifying login credentials

🔐 **Function: createOrUpdatePassword(string $loginadmin, string $plaintextPassword): bool**

- Checks if the loginadmin exists in the msadmin table:
  - If it exists → updates the password.
  - If it doesn't → creates a new admin account.
- Passwords are securely hashed using password_hash() (modern and secure PHP practice).

🔐 **Function: updatePassword(int $adminId, string $newPlaintext): bool**

- Updates the password for a given admin ID.
- Password is hashed before saving.
- Logs the action using Logger::info() or Logger::error().

🔐 Function: verify(string $loginadmin, string $plaintext): bool

- Checks if the username exists.
- If it does, verifies the password using password_verify().
- Returns true if valid, false if the username or password is incorrect.

💾 Database Table Used: msadmin

```
CREATE TABLE msadmin (
   adminid INT PRIMARY KEY,
   loginadmin VARCHAR,
   loginpassword VARCHAR
);
```

2.3.2. src/Database.php

This is a lightweight **PDO-based database abstraction class**, designed to:

- Prevent SQL injection,
- Simplify querying across your app,
- Provide a modern OOP interface to interact with MySQL.

🎯 Primary Goal:

Provide secure, consistent database access using **PHP PDO**.

✅ Methods:

- query($sql, $params = []): Executes a prepared statement and returns the PDOStatement.
- fetchAll($sql, $params = []): Returns all matching rows as an array.
- fetch($sql, $params = []): Returns the first row (or null if none).
- execute($sql, $params = []): Executes INSERT, UPDATE, or DELETE; returns the number of rows affected.

🔒 Security

100% protected from SQL injection using prepare + bind + execute.

2.3.3. src/index.php

This file is unused.
Its only purpose is to **prevent users from browsing** the src/ folder directly by default.

2.3.4. src/Logger.php

A simple and configurable logging class for writing messages to a log file.
It is useful for debugging, audits, and tracking application behavior—especially in production.

🎯 **Purpose:**

To log important application events with levels like:

- DEBUG
- INFO
- WARNING
- ERROR

⚙️ **Method: init(array $config)**

- Initializes the logger with:
  - path: file path for log file.
  - level: minimum log level to write.
- Automatically creates the log folder if it doesn't exist.

📄 **Method: log($level, $message)**

- Internal function that writes log messages in this format:

```
[2025-06-30 10:00:01][INFO] Admin login success
```

🔧 **Log Level Methods:**

Method	Description
debug()	Logged only if level is DEBUG
info()	Logged if level is DEBUG or INFO

Method	Description
warning()	Logged if level is DEBUG, INFO, or WARNING
error()	Always logged regardless of configured level

✅ **Logger Benefits**

Feature	Description
✅ Simple & fast	No third-party library needed
✅ Auto-create folders	Prevents "folder not found" errors
✅ Log level control	Allows lightweight logs in production
✅ CodeCanyon-ready	Clean and configurable logging preferred by reviewers

📝 **Sample Log Output:**

```
[2025-06-30 15:42:11][INFO] Admin 'kukuh' successfully logged in
[2025-06-30 15:42:15][ERROR] Failed to connect to Replicate API
[2025-06-30 15:42:17][DEBUG] Input: design_id=45, user_id=2
```

2.3.5. src/MasterDesign.php

This class manages **CRUD operations for master T-shirt design data** uploaded by users.

🎯 **Purpose:**

Handle backend management for mockup design files:

- Add new designs
- Retrieve all designs
- Delete designs (from DB and file system)

🔧 **Properties:**

- private Database $db: database access instance
- private string $uploadDir: the file path where designs are stored

🛠️ **Constructor: __construct(Database $db, string $uploadDir)**

- Initializes database connection and upload path.

- Automatically appends trailing slash (/) to the upload directory path.

✅ Method: insert(string $title, string $filename): bool

- Adds a new design to the masterdesign table.
- Sets default values:
  - titledesign: the title of the design
  - imagesdesign: uploaded file name
  - ispublish: default 1
  - submitdate: current date

Example:

$design->insert("Naruto T-shirt", "naruto123.png");

✅ Method: fetchAll(): array

- Returns all designs sorted by submitdate (newest first).

✅ Method: delete(int $id): bool

- Deletes the image file from disk and its record from the database.
- Steps:
  1. Fetch file name from database.
  2. Remove file from the folder (if exists).
  3. Delete record from DB.

Example:

$design->delete(5); // deletes design with ID 5

🛡️ Safety Features

Feature	Description
✅ Prepared Statements	Prevents SQL injection
✅ File existence check	Ensures file exists before deleting
✅ Silent unlinking	Uses @unlink() to prevent warnings

🗄️ Database Table: masterdesign

CREATE TABLE masterdesign (

```
    id INT AUTO_INCREMENT PRIMARY KEY,
    titledesign VARCHAR(255),
    imagesdesign VARCHAR(255),
    ispublish TINYINT DEFAULT 1,
    submitdate DATETIME
);
```

2.3.6. src/PersonModel.php

📄 **File: src/PersonModel.php**

📌 **Purpose:**

This class handles **CRUD operations (Create, Read, Delete)** for managing **person endorser images**, which are typically photo models used in AI T-shirt mockup applications.

🧱 **Class Structure: PersonModel**

🔧 **Properties:**

private Database $db;

private string $uploadDir;

- $db: Instance of the Database class used to run SQL queries securely via PDO.
- $uploadDir: The folder path where person images are stored on the server.

🔨 **Constructor:**

public function __construct(Database $db, string $uploadDir)

- Initializes the database connection and the upload directory.
- Ensures the directory path ends with a trailing slash / using rtrim().

✅ **Methods**

1. insert(string $title, string $filename): bool

```
$sql = "INSERT INTO person (titleperson, person_images, submitdate)
        VALUES (:title, :img, NOW())";
```

- Inserts a new person record into the person table.
- Parameters:
  - title: name or title of the person
  - filename: uploaded image file name
- NOW() sets the current date and time as the submitdate.
- Returns true if insert was successful.

---

2. fetchAll(): array

```
return $this->db->fetchAll('SELECT * FROM person ORDER BY submitdate DESC');
```

- Retrieves all records from the person table.
- Sorts by submitdate in descending order (latest first).
- Returns an array of person entries.

---

3. delete(int $id): bool

```
$row = $this->db->fetch('SELECT person_images FROM person WHERE id = :id', [':id'=>$id]);
```

- Deletes a person record by its id.
- Steps:
  1. Retrieves the filename of the image for the specified person.
  2. Checks if the image file exists in the upload directory.
  3. Deletes the file using @unlink() (suppresses warning if file not found).
  4. Deletes the record from the person table in the database.
- Returns true if deletion was successful.

🗃️ Database Table: person

```
CREATE TABLE person (
  id INT NOT NULL,
  titleperson VARCHAR(255) NOT NULL,
  person_images TEXT NOT NULL,
  submitdate DATETIME NOT NULL
);
```

🧾 **Fields:**

- id: Unique identifier for each person
- titleperson: The name/title of the person (model)
- person_images: The image filename
- submitdate: Timestamp when the record was added

✅ **Summary:**

PersonModel.php is a utility class to manage **photo models or person endorsers** in your T-shirt design application. It:

- Allows uploading new person images,
- Lists all uploaded persons,
- Deletes records and cleans up image files,
- Keeps your database and file system in sync.

The class uses **prepared statements** (safe from SQL injection) and **handles file deletion gracefully**.

2.3.7. src/PhotoSample.php

📄 **File: src/PhotoSample.php**

📌 **Purpose:**

This PHP class handles **CRUD operations** (Create, Read, Update, Delete) for **photo samples**, which are **combinations of a T-shirt design and a person (model)** image, along with the generated result using AI.

It is part of an AI-powered T-shirt design app that merges user designs with model photos to create realistic previews.

🧱 **Class: PhotoSample**

🔧 Properties

private Database $db;

- $db: A database instance (from the Database class) used to run SQL queries securely using PDO.

🔨 Constructor

public function __construct(Database $db)

- Initializes the class with a reference to the database connection.

✅ Methods

1. exists(int $designId, int $personId): bool

SELECT id FROM photosample WHERE designid = :d AND personid = :p

- Checks if a combination of a given design and person already exists in the photosample table.
- Returns true if a record exists, otherwise false.

---

2. updatePredictionId(int $id, string $newPredictionId): bool

UPDATE photosample SET predictionid = :pred WHERE id = :id

- Updates the predictionid of an existing record, based on its unique id.
- Returns true if the update was successful.

---

3. insert(...)

INSERT INTO photosample (...)

VALUES (:pred, :d, :p, :u1, :u2, :res, :isp, NOW())

- Inserts a new sample record into the photosample table.
- Parameters:
  - predictionId: The ID returned by the AI model (like from Replicate API).
  - designId: ID of the selected T-shirt design.
  - personId: ID of the selected model.
  - urlDesign: URL/path of the design image.

    - urlPerson: URL/path of the person image.
    - urlResult: URL/path of the AI-generated result (optional at insert).
    - publish: Whether to publish the result or not (1 = yes, 0 = no).
- Automatically sets the current time as generateddate.

---

4. updateResult(int $id, string $imageUrl): bool

UPDATE photosample SET imagesresult = :img, ispublish = 1 WHERE id = :id

- Updates the imagesresult column (the final generated image).
- Also sets ispublish to 1 (true).
- Returns true if the update was successful.

---

5. fetchAll(): array

*SELECT * FROM photosample ORDER BY generateddate DESC*

- Retrieves all photo sample records from the database, ordered by newest first.
- Returns an array of all photo samples.

---

6. delete(int $id): bool

DELETE FROM photosample WHERE id = :id

- Deletes a sample record by its id.
- Returns true if a row was successfully deleted.

🗃️ Related Database Table: photosample

Table Structure:

```
CREATE TABLE photosample (
  id INT NOT NULL,
  predictionid TEXT NOT NULL,
  designid INT NOT NULL,
  personid INT NOT NULL,
  urlimagesdesign TEXT NOT NULL,
  urlimagesperson TEXT NOT NULL,
  imagesresult TEXT NOT NULL,
  ispublish TINYINT(1) NOT NULL,
  generateddate DATETIME NOT NULL
);
```

Field Descriptions:

Column	Description
Id	Primary key (usually auto-incremented)
predictionid	Identifier of the AI-generated prediction
designid	Foreign key to the T-shirt design used
personid	Foreign key to the model (person) used
urlimagesdesign	URL or path to the design image
urlimagesperson	URL or path to the model photo
imagesresult	URL or path to the AI-generated result image
Ispublish	Whether this result is publicly visible (1/0)
generateddate	Timestamp of when the result was created

✅ **Summary:**

The PhotoSample class is the **core handler for combining design + model photos**, managing the resulting image generated by AI.

It provides:

- Checks for existing combinations
- Creation of new samples
- Updates of prediction and result images
- Deletion of records
- Retrieval of all photo samples for display

It works alongside the AI generation process (e.g., Replicate API) and is tightly integrated with your app's admin interface.

2.3.8. src/ReplicateService.php

📄 **File: src/ReplicateService.php**

📌 **Purpose:**

This PHP class is responsible for **interacting with the Replicate API**, a third-party image generation service. It sends requests to generate an image by merging two input images with a prompt and optionally waits for the result to be ready.

Used in an AI-powered T-shirt design application, this class helps **merge the T-shirt design and model photo** into a realistic AI-generated image.

🧱 **Class: ReplicateService**

🔧 **Property**

private string $token;

- Stores the API token for authenticating requests to the Replicate API.

🔨 **Constructor**

public function __construct(string $token)

- Initializes the service with the API token needed to make authorized calls to Replicate.

✅ **Methods**

---

1. generate(string $input1, string $input2, string $prompt): array

🔹 **Purpose:**

Sends a request to the Replicate API to **create a new image generation job (prediction)**.

📥 **Parameters:**

- input1: URL or base64 of the first image (e.g., design)
- input2: URL or base64 of the second image (e.g., model)
- prompt: A text instruction guiding how the AI should combine the images

📤 **Returns:**

- An array of data returned by the Replicate API, including the prediction ID

🧠 **Internal Notes:**

- Uses curl to send a POST request
- Uses the Prefer: wait header to signal that it prefers a synchronous response
- Throws a RuntimeException if the API returns an error

---

2. getResult(string $predictionId, int $retries = 10, int $delay = 8): array

🔹 **Purpose:**

Polls the Replicate API using a predictionId to check if the image generation is complete.

📥 **Parameters:**

- predictionId: The ID received from the generate() method
- retries: How many times to retry (default 10)
- delay: Delay in seconds between each retry (default 8)

🔁 **Behavior:**

- Sends up to retries HTTP GET requests to Replicate
- If the output is ready (i.e., output key exists), it returns the result
- If no output is ready after all retries, throws a RuntimeException

---

3. generateAndWait(...): Combination of Generate + Polling

🔹 **Purpose:**

A **high-level function** that:

1. Starts the generation request,
2. Waits until the output is ready by polling,
3. Logs progress using Logger

📥 **Parameters:**

- input1, input2, prompt: Same as generate()

- retries, delay: Control the polling behavior

🔧 Workflow:

generate() → get prediction ID → poll result → return output

🔒 Error Handling:

- Logs every polling attempt
- Logs when the output is ready
- Throws a clear exception if the image is not generated after all retries

---

🔐 Example Use Case

Used when a user uploads a design and selects a model photo. This class:

- Sends the combination to Replicate with a description (prompt)
- Waits for the result image to be ready
- Saves the result to be displayed or published

🔧 Logging

The class uses Logger::info() and Logger::debug() to log:

- When generation starts
- Each polling attempt
- When the output is ready
- If retries fail

✅ Summary

ReplicateService.php is a utility class that:

- Sends image generation jobs to the Replicate API
- Supports both synchronous and asynchronous workflows
- Includes built-in retry and logging mechanisms
- Forms the **core integration between your PHP backend and Replicate's AI image generator**

It's essential for generating mockup previews by combining design and model images.

2.3.9. src/Settings.php

📄 **File: src/Settings.php**

📌 **Purpose:**

This class provides a simple, database-driven **configuration management system**. It allows you to **get** and **set** key-value pairs in a settings table, such as the API token for Replicate or the base URL of the website.

It is useful for storing environment variables or dynamic configuration values that can be changed without editing the PHP code.

🧱 **Class: Settings**

🔧 **Property:**

private Database $db;

- Holds the reference to the Database class for executing SQL queries securely using PDO.

🔨 **Constructor:**

public function __construct(Database $db)

- Initializes the class with the given database connection.

✅ **Methods**

---

1. get(string $key): ?string

$row = $this->db->fetch('SELECT `value` FROM settings WHERE `key` = :key', [':key'=>$key]);

return $row['value'] ?? null;

🔹 **Purpose:**

- Retrieves the value for a given configuration key.
- Returns the value as a string if found, otherwise null.

📥 Example Usage:

```
$token = $settings->get('REPLICATE_API_TOKEN');

$host = $settings->get('HOST_DOMAIN');
```

---

2. set(string $key, string $value): bool

```
$row = $this->db->fetch('SELECT id FROM settings WHERE `key` = :key', [':key'=>$key]);

if ($row) {

    return $this->db->execute('UPDATE settings SET `value` = :value WHERE `key` = :key',
[':value'=>$value,':key'=>$key])>0;

}

return $this->db->execute('INSERT INTO settings (`key`,`value`) VALUES (:key,:value)',
[':key'=>$key,':value'=>$value])>0;
```

🔹 Purpose:

- **Updates** an existing key if it already exists.
- **Inserts** a new key-value pair if it does not exist.
- Returns true if the insert/update was successful.

📥 Example Usage:

```
$settings->set('REPLICATE_API_TOKEN', 'your_token_here');

$settings->set('HOST_DOMAIN', 'https://yourdomain.com/app');
```

🗃️ Database Table: settings

Table Structure:

```
CREATE TABLE `settings` (

  `id` INT NOT NULL,

  `key` VARCHAR(255) NOT NULL,

  `value` TEXT NOT NULL

);
```

Field Descriptions:

Column	**Description**
id	Primary key (usually auto-incremented)
key	The configuration key (e.g., REPLICATE_API_TOKEN)
value	The configuration value (e.g., the API token or domain URL)

📦 **Sample Data:**

INSERT INTO `settings` (`id`, `key`, `value`) VALUES

(1, 'REPLICATE_API_TOKEN', '........'),

(2, 'HOST_DOMAIN', 'https://yourdomain.com/yourfolder/public');

This sets:

- Your API token for image generation,
- The base domain used for constructing full image URLs or routing.

✅ **Summary:**

The Settings class is a **lightweight configuration manager** for your PHP application. It enables you to:

- Store and retrieve key settings like API tokens or host URLs from a database,
- Avoid hardcoding sensitive or environment-specific values in your code,
- Easily update config values from the database without redeploying your app.

It works well alongside the ReplicateService and other components that need dynamic settings.

2.4.1. bootstrap.php

🔹 **Purpose:**

This file acts as the **entry point (bootstrap)** for your PHP application. It:

- Loads all core class files
- Loads the configuration
- Initializes the logger

- Establishes a database connection
- Instantiates helper classes like AdminAuth and Settings

🔤 Code with English Comments:

```php
<?php
/**
 * bootstrap.php
 * Application bootstrap: loads config, initializes logger & database
 *
 * 🛠️ AI T-Shirt Design Application
 * Created by: Kukuh TW
 */

// 1. Load core class files
require __DIR__ . '/src/Logger.php';
require __DIR__ . '/src/Database.php';
require __DIR__ . '/src/AdminAuth.php';
require __DIR__ . '/src/MasterDesign.php';
require __DIR__ . '/src/PersonModel.php';
require __DIR__ . '/src/PhotoSample.php';
require __DIR__ . '/src/ReplicateService.php';
require __DIR__ . '/src/Settings.php';

// 2. Load configuration file
$config = require __DIR__ . '/config.php';

// 3. Initialize logger
Logger::init($config['logger']);
Logger::info('Logger initialized.');

// 4. Initialize database connection
try {
    $db = new Database($config['db']);
    Logger::info('Database connection established.');
} catch (Exception $e) {
    die('Fatal error: ' . $e->getMessage());
}

// 5. Initialize Admin Authentication and Settings classes
$adminAuth = new AdminAuth($db, $config['features']['admin_password']);
$settings  = new Settings($db);
?>
```

2.4.2. config.php

🔹 **Purpose:**

This file contains the application's **configuration settings**, including:

- Database credentials
- Logger configuration
- Feature flags

🔤 **Code with English Comments:**

```
<?php
/**
 * config.php
 * Configuration settings for the application
 *
 * 🛠️ AI T-Shirt Design Application
 * Created by: Kukuh TW
 */

return [
    'db' => [
        'host'     => 'localhost',       // Database host
        'dbname'   => 'designkaos',      // Database name
        'user'     => 'root',           // Database username
        'password' => '',               // Database password
        'charset'  => 'utf8mb4',        // Character encoding
    ],
    'logger' => [
        'path'  => __DIR__ . '/logs/app.log',  // Log file path
        'level' => 'DEBUG',                   // Minimum log level (DEBUG, INFO, etc.)
    ],
    'features' => [
        'admin_password' => false,  // Set to true only once to create admin password; set back to false after setup
    ],
];
?>
```

2.4.3 .htaccess

🔹 **Purpose:**

This file defines **URL rewrite rules** using Apache's mod_rewrite. It ensures:

- When a user visits /yourfolder/, they are automatically redirected to the login page at /yourfolder/public/login.php

🔤 **Rewritten Explanation in English:**

```
RewriteEngine On
RewriteBase /yourfolder/

# If the URL is just /yourfolder or /yourfolder/, redirect to the login page
RewriteCond %{REQUEST_URI} ^/yourfolder/?$ [NC]
RewriteRule ^$ public/login.php [L,R=302]
```

🔁 **Note:** Replace /yourfolder/ with the actual folder name where your app is located, such as /designkaos/.

✅ **Summary**

File	**Purpose**
bootstrap.php	Initializes the core system (classes, logger, database, auth)
config.php	Central place for app configuration like DB connection and logging
.htaccess	Sets URL rewriting rules and redirection for default routing behavior

This setup ensures a clean structure for initializing, securing, and routing your PHP web application.

👤 Developer Contact

- **Name:** Kukuh TW
- **Email:** kukuhtw@gmail.com
- **WhatsApp:** +628129893706
- **Instagram:** @kukuhtw
- **Twitter/X:** @kukuhtw
- **Facebook:** facebook.com/kukuhtw
- **LinkedIn:** linkedin.com/in/kukuhtw