worldskillsuk

WSUK – National Finals Competition

Web Development

2025

MODULE B

BACKEND DEVELOPMENT

Val Adamescu
Lewis Newton
Mark Kiss

Contents

Introduction

In this module, you are required to build a site that can be used by an administrator to manage key aspects of a photography business. You can write your solution using both front- and back-end technologies, including HTML, CSS, JavaScript, PHP, and SQL. You can also use any of the provided frameworks, should you wish.

The test project can be completed without using any framework; it is your decision whether you use one. If you do use any of the frameworks, the submitted solution must be functional as-is; no attempts will be made by the judges to fix or manipulate the code in any way. For any technical documentation you can use the standalone or browser enabled *DevDocs.io*.

If you are using any framework, you must provide a basic READ.me of the initialisation (e.g. `npm run dev`, `npm start` etc) or any other instruction to ensure that the judges will be able to run your project.

NOTE: *The specific submission/upload requirements of the test project are provided separately.*

You have three (3) hours to complete this task. You need to submit it before time runs out. No additional time will be given for submission.

Requirements

Spark Studios, a new photography and digital design agency in Wales, needs an efficient way to manage customer print orders from events they photograph. Your task is to build a bespoke, internal ***Administration System*** that allows authorised users to manage **events**, **pictures**, **orders**, and **settings** through a secure dashboard. The system must also provide limited **public access** to selected events via shareable links, enabling customers to view and download images associated with their event.

Some reference designs are provided in the `dist/templates` folder so your focus will be on implementing functionalities rather than produce designs. Each template is an individual HTML/CSS page, and you can use/follow these designs, or you can make your own. This module, is backend focused hence the design is provided – the webapp still need to have a minimum aesthetic. For more information look at `dist/templates/README.md`

Core Requirements:

The system supports two entry paths: administrators and public visitors. Administrators authenticate securely to manage events, pictures, orders, and reusable settings through a protected dashboard, while the public can browse open events or use access codes to view private ones. All data operations — creation, updates, uploads, and deletions — occur through validated forms and server-controlled routes. Pictures and orders remain linked to their respective events, ensuring consistent data relationships and visibility control. The backend maintains strict separation of privileges, secure session handling, and a seamless transition between public and administrative contexts.

Administrators should be able to:

- Log in and out securely, maintaining session state.
- View key statistics from orders and uploaded pictures.
- Create, edit, and delete events, pictures, and orders.

- Manage reusable settings such as categories, picture sizes, and user accounts.
- Generate and manage access codes for shareable event views.

The public (unauthenticated) should allow visitors to:

- Access public events.
- With a valid access code access for a specific private event.
- View event details and associated pictures once access is granted.
- The backend must therefore handle secure role separation, proper data relationships, and consistent user experience across both administrative and public contexts.

General

- **Database:** A workable database ERD is given to you with all relationships. *Payment processing is handled by third-party services and is not required.*
- **Navigation:** You must implement the specific URL routes defined in the brief.
- **Design:** The site does **not** need a beautiful design, but it must be user-friendly. Minimal page designs (wireframes) are provided as a guidance. You can use them as they are, or you can enhance the design.
- **Technology:** You are free to choose the technology stack, as long as it meets the requirements.

★Navigation and Routes

Routes cover both paths: public/guest pages and protected admin areas. All access is enforced server-side, with clean URLs, query-string pagination/sorting, and consistent layouts. After create/update/delete, users are redirected to the relevant listing with a clear status message; admin menus highlight the current section.

All routes must follow consistent URL patterns and enforce authentication, authorisation, and visibility rules on the server side, regardless of the chosen framework.

The application must be accessible from the base URL `/` andno other prefix such as `/public` prefix should be present.

The routes and endpoints are also present in `dist/ROUTES.md` and `dist/routes.yaml`

Route Audiences

Audience	Purpose
Admin	Authenticated administrators who manage `events`, `event access codes`, `pictures`, `orders`, and `system settings`.
Guest	Unauthenticated visitors who gain **temporary access** to `private events` using a valid event access code. Guests do not have accounts or sessions and can only view the specific event linked to their code.
Public	General visitors who can freely browse public events without authentication or access codes.

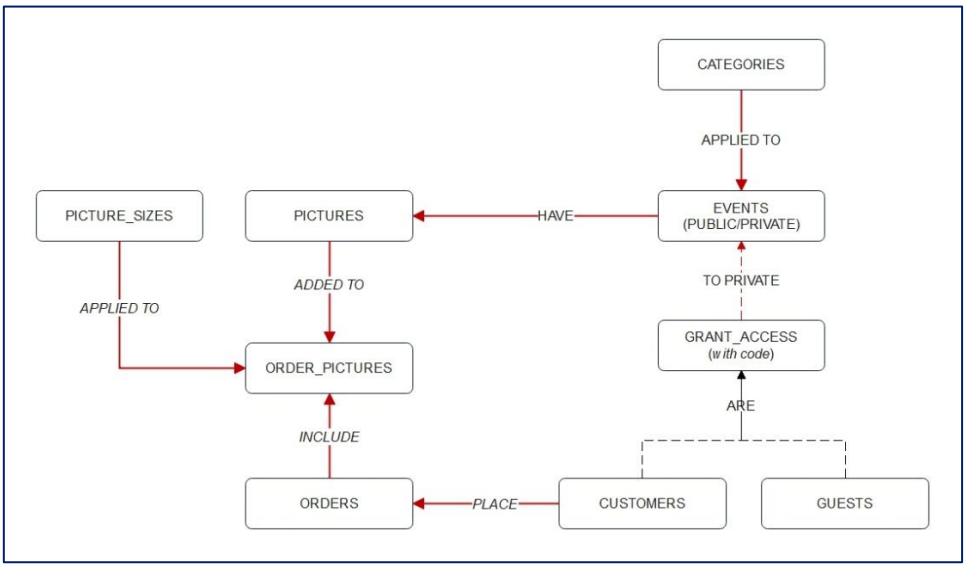
Generic Route Specification

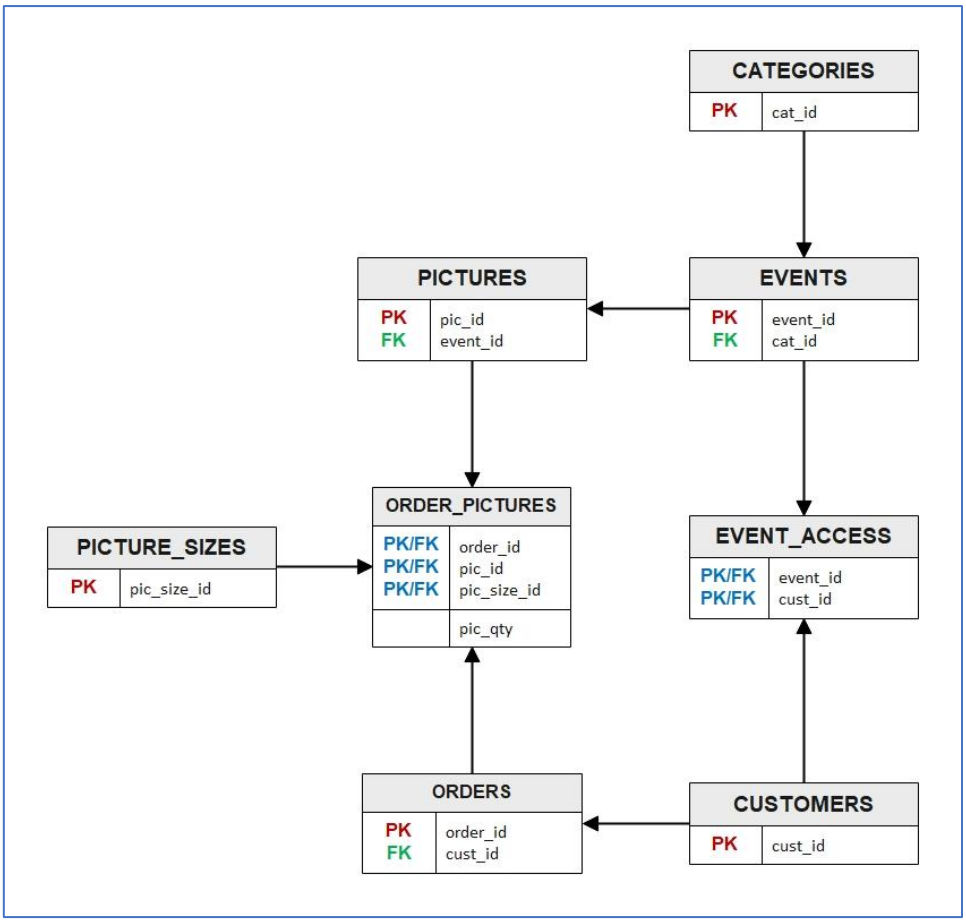
PATH	AUDIENCE	METHODS	PURPOSE/EXPECTED BEHAVIOUR
/	Admin / Guest	GET	Entry point – if authenticated, renders the Dashboard; if not, redirects to **/login**.
/logout	Admin	POST	Ends current session → redirects to **/login**.
/settings	Admin	GET	Opens the Settings hub with links to **Categories**, **Picture Sizes**, and **Users**.
/settings/categories	Admin	GET, POST, PUT, DELETE	Manage picture categories (list, create, update, delete).
/settings/picture-sizes	Admin	GET, POST, PUT, DELETE	Manage available picture sizes (list, create, update, delete).
/settings/users	Admin	GET, POST, DELETE	List admin accounts, create new admin, delete another admin.
/settings/users/{id}	Admin	DELETE	Delete a user account
/pictures	Admin	GET, POST, PUT, DELETE	Manage pictures (list, upload, edit, delete).
/events	Admin	GET, POST, PUT, DELETE	Manage events (list, create, update, delete).
/events/{id}/access	Admin	GET, POST	Generate or manage access codes for an event.
/events-access/{id}	Admin	PATCH, DELETE	Toggle or revoke an existing access code.
/orders	Admin	GET, PUT, DELETE	View and manage orders.
/orders/{id}	Admin	GET	Display order details (including pictures and totals).
/login	Guest	GET, POST	Dual-entry page: left panel = admin login (success → / dashboard); right panel = guest access-code form (submits to /verify-access-code).
/verify-access-code	Guest	POST	Validates access code → on success redirect to ***/events/private_events/{event_name}?access_code={code}***; on failure return to /login with error.
/public-events	Public	GET	Lists all public events.
/events/public_events/{event_name}	Public	GET	Displays a specific **public event** (e.g. marathon_finish_line); no access code required.
/events/private_events/{event_name}	Public	GET	Displays a **private event** for guests with a valid access_code; invalid or missing code → error or redirect to /login.
/events/{path}	Public	GET	Fallback for nested or shareable event URLs; must still enforce visibility and permissions.

- **Active Highlight** – The current route or route group must be visibly highlighted in the admin sidebar or navigation bar.
- **Layout Consistency** – All admin pages use a shared layout component for header, sidebar, and footer.
- **Visibility** – Guest and public pages must exclude admin-only links.
- **Pagination & Sorting** – First page shows by default; moving between pages must refresh results for that page number (server-side).
    - (e.g. ***/events?page=2, /orders?sort=date&direction=desc***).
- **Redirects & Feedback** – After ***create/update/delete*** operations, the system redirects back to the relevant listing and displays a status message (success or error).

★ Database

The following diagram is a representation of the core entities and relationships. You can add new relationships if you wish has long the requirements are fulfilled, and the database normalisation is in the **3NF.**





You will need ***to optimise*** the provided basic SQL schema (`database/module-b-schema.sql`) by creating additional attributes, allocate the ***correct and optimised data type/size*** and *create PKs/FKs referential integrity at the database level*.

★ Login Page

The login page splits access between administrators and general visitors. Admins log in with email and password to reach the dashboard, while guests can either browse all public events directly or enter an

access code to view a specific private event. Both options are validated and redirect users to the right section automatically.

Requirements

- The login page must be accessible at ***/Login***.
- The page must contain two forms/panels:
  - **Admin Login** – allows administrators to sign in using email and password.
  - **Guest Access** – allows guests to navigate to the public events or enter an access code to access a private event
- Both panels must be displayed on the same screen (see ***dist/templates/login.html***).
- Submitting valid admin credentials must start a session and redirect to /.
- Submitting a valid access code should redirect to a private event that had associated code e.g: ***/events/private_events/{event_name}?access_code={code}***.
- Invalid credentials or access codes must keep the user on the same page and display an error message.
- Logged-in administrators attempting to access /login must be redirected to /.
- Both forms must include full front-end validation for required fields.
- Back-end validation must return specific error messages for failed logins or invalid access codes.
- The interface should remain responsive on all screen sizes.
- Only one request (***Login*** or ***access code***) may be submitted at a time.
- On successful login or access-code validation, the user is redirected automatically to the correct route without manual refresh.

★ Authentication

Authentication verifies admin login details and manages active sessions. Only logged-in admins can access protected routes, while unauthorised users are redirected to the login page.

Requirements

- Admin authentication must be handled through the /login route.
- Login form with mandatory fields: ***email*** and ***password***
- Front-end validation must ensure fields are not empty and email is in valid format.
- Back-end validation must verify credentials and return specific error messages for invalid or missing data.
- On successful authentication:
  - Start a new session.
  - Redirect to **/** (***dashboard***).
- On authentication failure:
  - Stay on the same page.
  - Show an error message ("*Invalid email or password*").
- Logged-in administrators attempting to access ***/Login*** must be redirected to **/**.
- Logging out must send a ***POST*** request to ***/Logout*** and immediately end the session.
- After logout, redirect to ***/Login***.

- All protected routes (***dashboard***, ***events***, ***pictures***, ***orders***, ***settings***) must only be accessible to authenticated users (administrators).
- Unauthenticated access to any admin route must redirect to ***/Login***.
- Session management must persist until logout; refreshing the browser must not invalidate an active session.
- Only administrators exist as users. No public registration is required.
- Administrators may create other admins

Behaviour

- The authentication system must be fully server-side and not rely on client-side route checks.
- Session data must be securely stored and cleared on logout.
- Attempting to access guest-only or public routes (e.g. ***/Login***, ***/public-events***) while authenticated must redirect to **/**.
- Error and success messages must be displayed clearly and remain consistent across all authentication-related views.

★ Authorisation

Authorisation controls what each user can access. Admins have full system permissions, while public visitors can only view public pages or private events with a valid access code (see events management).

Requirements

- Only **administrators** exist as system users, *no customer account(s)*.
- **Protected routes** (admin only): **/**, ***/events*****, ***/pictures*****, ***/orders*****, ***/settings*****.
- **Guest routes**: ***/Login***, ***/verify-access-code***.
- **Public routes**: ***/public-events***, ***/events/public_events/{event_name}***.
- **Private event routes**: ***/events/private_events/{event_name}?access_code={code}***.
- **Access-code logic**:
    - A guest with a valid code is **authorised** to view only the specific event that the code was generated for.
    - Codes must be validated server-side and linked to an active event.
    - Invalid or disabled codes will deny access and redirect to ***/Login*** or show an error.

Behaviour

- Unauthenticated request to admin ***route → redirect*** to ***/Login***.
- Authenticated admin visiting guest***/public route → redirect to*** /.
- Guest with valid ***access_code*** → authorised view of the code associated private event only.
- Authorisation checks must run on the server; client-side visibility changes alone are not sufficient.

★ Public Events

The public events area lets anyone browse open events without needing an account. Each event shows key details like its name, category, city, and date, with a link to view photos. Everything shown here comes directly from active records in the database, so only approved and up-to-date events appear to the public.

Requirements

- Route: ***`GET /public-events`***.
- Page must show **title** "*Public Events*" and a short subtitle "*Browse our upcoming public photography events*".
- A **Back to Login** button must link to ***`/login`***.
- Only ***`active`*** events with visibility ***`public`*** must be listed by date ***`DESC`*** – newest event first.
- Only ***`active pictures`*** for event will be displayed (see pictures section)
- Grid layout: **cards** in a responsive 5-column grid for 1920px (collapse appropriately on tablet/mobile).
- Each card must display:
  - **Event Name**
  - **Category badge**
  - **City** of event
  - **Date**
  - **Time**
  - **Note** (show "–" if empty)
  - **CTA button** "*View Event Photos*"
- CTA target: /events/public_events/{event_name} or {id}.

Behaviour

- Cards must navigate correctly to their event detail URL.
- If no public events exist, show a clear "*No public events available*." message.
- Page must remain accessible without authentication and must not expose admin navigation.

★ Event Preview (Public / Private)

The event preview page displays full details and pictures for a selected event. Public events open freely, while private ones require a valid access code (generated by admin – see 'Events Management' section) before loading their content.

Requirements

- Route:
  - Public: ***`GET /events/public_events/{event_name}`***
  - Private: ***`GET /events/private_events/{event_name}?access_code={code}`***
- The same page layout and design should be used for both public and private event previews. A template is available in ***`dist/templates/public-event.html`*** – you can modify if you wish so.
- For ***`public events`***, the page is directly accessible without validation.
- For ***`private events`***, a valid ***`access_code`*** must be provided and verified server-side.
  - If the code is missing or invalid, show an access error message.
  - Create at least a private event with the name '***`Private-Test-Event`***' for testing (exact naming)
- Page content must include:
  - **'***Public Events***'** as the title and the subtitle '*Browse our public photography events*'
  - Back to Public Events ***`button`*** → ***`/public-events`***.

- Each *event card* (or any other layout) must contain
  - Event title (full name).
  - Category badge
  - Location (city)
  - Date & Time (formatted ***D Month, YYYY HH:MM***)
- **Access banner:**
  - For private events: "*Access granted on {timestamp}.*"
  - Hidden for public events.
- **Event Photos** section displaying all active pictures in a responsive grid.

- If no pictures are available, display "*No photos available for this event yet.*"
- The ***{event_name}*** in the URL should be an SEO-friendly slug.

Behaviour

- Page accessible to both guests (public) and verified access-code holders (private).
- Refreshing or sharing the valid private event URL (with code) must reopen the same view.
- If a ***public event*** is marked as ***inactive***, it must ***not be displayed*** on the public events list ***and must not be accessible*** even through a direct URL or event ID.
- If a ***private event*** is marked as ***inactive***, the user must be shown a clear message stating that ***the event has been disabled by an administrator***, and ***no access will be granted***, even if a valid access code is provided.
- Page must not show admin navigation or dashboard elements.

★ Dashboard

The dashboard gives admins an overview of system activity. It displays key stats from events, orders, and pictures, with links to manage each section directly.

Requirements

- Route: ***GET /*** (authenticated only). Unauthenticated → redirect to ***/Login***.
- Header shows current admin name and a ***Logout*** button.
- Sidebar links: ***Dashboard***, ***Events***, ***Pictures***, ***Orders***, ***Settings*** (Dashboard highlighted).
- A ***Create Event*** button must be visible on the dashboard.
- Show minimum four **statistic cards**, all **calculated on the back end**: Below are some examples of the KPIs but *you can create your own*
  - **Orders** — "*X of Y completed*" and "*N new this week*"; include a progress bar for completion ratio.
  - **Avg. Order Picture Count** average number of pictures per order on a weekly basis, monthly or overall
  - **Popular Picture Sizes** — list picture sizes with orders counts.
  - **Income by Picture Size** — list sizes with totals.
- At least one KPI should have a bar, graph, pie chart or any other visual element; not just a number
- Card titles, values, or bars must match database values at render time - not hard coded but calculated from DB values.
- Links within cards (where applicable) must navigate to their corresponding pages.

Behaviour

- Page uses a shared admin layout, and ***Dashboard*** is the active route.
- Data updates after create/edit/delete actions elsewhere and reflects current DB state on reload.
- Currency values use ***GBP (£)*** formatting.

★ Events – Management

The events section allows admins to create, edit, or delete event records. Each event includes basic details like name, type, date, and city, with private events automatically generating access codes for guest viewing.

Requirements

All events should be listed in a table with the following information available

- **Columns:** ***Event Name***, ***Category***, ***Type (Public/Private)***, ***City***, ***Date/Time***, ***Note***, ***Event Path***, ***Actions (Edit***, ***Delete***, ***Access, Active/Inactive).***
- **Sorting:** default ***Name ASC***; toggles for ***Name***, ***City***, ***Date & Time*** (ASC/DESC).
- ***Create/Edit*** form fields**:** ***name***, ***category*** (select), ***type*** (public/private), ***city***, ***datetime***, ***note*** (optional) – template available in ***dist/templates/forms.html***.
- **Path/slug:** auto-generated from name; prefix ***public_events/*** or ***private_events/*** by type; show preview on form.
- **Validation:** required fields validated on front end and back end; datetime must be valid; note optional.
- **On success:**
  - ***Create*** → insert row → redirect to ***/events*** with status message.
  - ***Update*** → persist changes (including slug if name changed) → redirect to ***/events*** with status message.
  - ***Delete*** → remove row → redirect to ***/events*** with status message.
- **Access codes** (*for private events only*) under ***Access*** button in actions:
  - List codes with ***code***, ***status*** *(active/revoked)*, ***created/last used***.
  - Actions: ***Create***, ***Toggle Active***, ***Delete***.
  - Generated code must be unique across all events and return a shareable URL: ***/events/private_events/{slug}?access_code={code}***.
  - The access codes must be **a minimum of 8 characters long**, containing only **uppercase and lowercase letters (*A–Z, a–z*)** and **numbers (*0–9*)**.
  - All codes must be ***encrypted*** **before being stored** in the database.
  - Create at least a custom code '***TestCode123***' for '***Private-Test-Event***' (exact naming and capitalisation)

Behaviour

- Clicking sortable headers updates query params and refreshes the table.
- "*Create Event*" opens the create form/page; "*Edit*" opens a pre-filled form; "*Delete*" requires confirmation.
- For ***public*** events, the ***Access*** action is disabled/hidden

- Only ***active*** events appear in ***public views***; admin can manage all events here.
- All operations surface clear ***success/error messages*** and never expose admin navigation on public pages (*see Security section*).

★ Pictures – Management and Upload

The pictures section lets admins upload, edit, or delete images linked to specific events. Each upload is validated for type and size, stored securely, and can be toggled active or inactive to control what appears on public event pages.

Requirements

- **Main page:**
  - Responsive card grid with one Picture ***Preview,*** Event ***name*** (linked to related event), ***Event Category***, ***Upload Date, File Path, Status*** badge (***Active*** / ***Inactive***) and ***actions***. Each column must have a clear label (***PREVIEW***, ***EVENT NAME***, ***EVENT CATEGORY***, ***SUPLOAD DATE, STATUS, ACTIONS***).
  - Button: ***Upload New Picture*** → ***/pictures/create***.
- **Upload page** supports ***drag-and-drop*** area and ***multi-select file picker*** methods:
  - Title: "***Bulk Picture Upload***"
  - **Individual:** select one image, or multiple by holding SHIFT key
  - **Supported files:** ***.jpeg***, ***.jpg***, ***.png***; **max 5 MB each**.
  - **Per-image required metadata: Event** (select); **Note** optional.
- **Storage & paths:** save files under ***/storage/...***; persist relative path in DB. Visibility (***public/private***) must not affect the naming scheme and the file.
- **Validation:** server- and client-side checks for required fields, file type, and size. Invalid files must return info errors; valid files continue.
- **On success:** insert DB rows with ***filename***, ***event/category***, ***note***, and ***is_active=1***; then redirect to ***/pictures*** with confirmation.

Behaviour

- All uploads should store file paths under to be accessible only by the web
- A picture can be made active/inactive for an event
- Only ***active*** pictures appear in public/event previews, status toggle available on Edit.
- Edit form allows toggling status (Active / Inactive) and updating note or associations.
- Deleting a picture removes the DB row and the stored file
- After creating, edit, or delete actions, ***/pictures*** view refreshes with updated grid.

★ Orders

The orders section lets admins review and update customer print requests. Each order lists its items, quantities, and prices, with status changes between pending and completed handled directly in the dashboard.

Requirements

- **List view columns:** ***`Order ID`***, ***`Customer Name`***, ***`Order Date`***, ***`Completed Date`*** (*nullable*), ***`Note`***, ***`Total Order`***, ***`Status`*** (***`Pending`*** / ***`Completed`*** / C***`ancelled`***) actions buttons or any other way of actions.
- **Default sort:** by ***`Order ID ASC`***. Provide sorting (***`ASC/DESC`***) function for ***`order id`***, ***`Order date`***, ***`Completed date`***, ***`status`***.
- **Row actions by status:**
  - **Pending:** ***`Complete`***, ***`Cancel`***, ***`Delete`***
  - **Completed:** ***`Mark as Pending`*** *(reverse to Pending Status)*, ***`Cancel`***
  - **Cancelled:** ***`Mark as Pending`*** *(reverse to Pending Status)*, ***`Delete`***
- **Status rules:**
  - ***`complete`*** → set status=completed, set ***`completed_at=now()`***.
  - ***`pending`*** → set status=pending, set ***`completed_at=NULL`***.
  - ***`cancel`*** → set status=cancelled, set ***`completed_at=NULL`***.
  - Ignore no-op transitions (e.g., ***`completing an already completed order`***).
- **Order Details page (/orders/{id}):**
  - Show ***`Order #`***, ***`customer name`***, ***`order date`***, ***`order status`***, ***`notes`*** (*if any*) and order items (*pictures in the order*):
  - For each item in the order: ***`Picture`*** **(*`thumb/id`*)**, ***`Picture Size`***, ***`Unit price`***, ***`Quantity`***, ***`Order total.`***
- **Data source:** all order and item data must be read from the database.
- **After any action:** redirect back to ***`/orders`*** (or the details page after save) with a visible success/error message.

Behaviour

- All monetary values must render in ***`GBP (£)`*** with 2-decimal formatting.
- Links from the list open the correct details page; returning to the list preserves the last used page/sort.
- Deleting an order removes the order and its items.
- Only admins can access orders; unauthenticated requests redirect to ***`/login`***.

★ Settings

The settings hub centralises reusable data: **Categories**, **Picture Sizes**, and **Admin Users**. Admins add/edit/delete categories (unique, case-insensitive) and define sizes (label, width, height, price), then manage users (create, delete). All forms use server-side validation and redirect back to their lists with clear status messages.

Requirements

Main Page

- Route: ***`GET /settings`***.
- Show three links/cards: ***`Manage Categories`***, ***`Manage Picture Sizes`***, ***`Manage Users (Admins)`***.

Categories

- List columns: ***`Category Name`***; actions: ***`Edit`***, ***`Delete`***.
- ***`Create/Edit`*** form: single ***`Category name`*** (required, non-empty, unique case-insensitive).
- After create/update/delete - redirect back to ***`/settings/categories`*** with status message.

Picture Sizes

- List columns: ***Label***, ***Width***, ***Height***, ***Price***; actions: ***Edit***, ***Delete***.
- Create/Edit fields (all required):
  - ***Label*** (text) | ***Width*** (positive integers) | ***Height*** (positive integers) |***Price*** (***DECIMAL, 2 > 0***)
- Validation shows info if errors.

Users (Admins)

- Create the following two users (you can create more but these are compulsory for tests)

User Name	Email	Password
Admin User1	admin1@admin.com	Password1
Admin User2	admin2@admin.com	Password2

- Routes:
  - **List/Create**: ***GET /settings/users***, ***POST /settings/users***
  - **Delete**: ***DELETE /settings/users/{id}***
- Create fields:
  - ***Name*** (required, string)
  - ***Email*** (required, valid, unique)
  - ***Password*** (required, confirmed)
- Passwords must be **hashed** before storage.
- After create/delete/status change: redirect back to /settings/users with status message.
- Newly created admin must be able to log in at ***/Login***.

Behaviour

- Forms use client-side required attributes plus server-side validation; errors display inline.
- Deleting a record must require confirmation; operations must not break referential integrity (block or validate before delete).

★ Security

Security is enforced end-to-end: passwords and access codes are hashed, inputs are validated both client- and server-side, and all queries use parameter binding. File uploads are checked for type/size and stored safely; errors are generic to avoid leaks. Protected routes and access-code checks run server-side, and sessions are regenerated/cleared on login/logout.

Requirements

- **Passwords** must be securely ***hashed before storage***. No plain text or reversible encryption.
- **Access codes**: minimum ***8 characters, [A-Z a-z 0-9]*** only, and ***stored encrypted***. Verification must compare against the encrypted value.
- **Input validation**: All form inputs must include ***client-side validation*** for common checks (required fields, format, length) and ***server-side validation*** for security and data integrity. The server must reject invalid or missing data and return informative error messages without losing previously entered values.
- **SQL protection**: all database queries must be written to prevent SQL injection by using prepared statements or the framework's built-in parameter binding instead of concatenating user input directly into SQL strings.
- **Authorisation on server**: protected routes require an authenticated admin; private events require a valid access code - (*see Event Preview and Events Management sections*);

- **Authentication flows**:
  - Login **POST** only; **logout POST** only.
  - **Regenerate session ID** on login and logout.
- **File uploads** (pictures): enforce type/extension checks (.jpeg, .jpg, .png), ***max size***, (5MB)
- **Error handling**: return ***generic messages***; do not expose stack traces, SQL, or filesystem paths to users.

Behaviour

- On validation failure, ***stay on the same page*** and show informative errors; preserve inputs where appropriate.
- Inactive events (public or private) must ***deny access*** with a clear message.

General

Requirements

- **Status messages** (success/error) must be shown prominently on all user-facing pages.
- **Failed back-end validations/actions** must return to the previous page with error messages and preserved inputs.

Behaviour

- After ***create/update/delete***, redirect back to the relevant listing with a success message.

On unexpected errors, show a generic error message without exposing stack traces or secrets.

Code Quality & Organisation

You are required to demonstrate good backend development practices by writing **clean, structured, and reusable code** that is easy to follow and maintain. All files, folders, and functions must use **clear, descriptive names** that reflect their purpose and follow a logical project structure. Repeated logic should be centralised into reusable functions or modules rather than duplicated (DRY).

Use brief comments only where needed and maintain consistent formatting, you should use GIT commits showing a clear development history rather than large unrelated changes and the repository must remain organised and easy to follow.

Media & Data Files

- ***templates***: A set of ***templates*** are provided as individual pages using HTML/CSS and minimal JS ***dist/templates*** folder.
- ***Data***: Dummy data to be used are provided in the ***dist/dummy_data*** folder. The data are provided in 3 formats as ***.csv***, ***.JSON*** and ***.txt***. All data are the same and you can use the most convenient format for you, or you can generate your own data.
- ***Test Files***:
  - 4/6MB ***.jpg***, ***.jpeg***, ***.png*** files
  - 4MB ***.TXT*** file to test other type of uploads
- ***Assets***:

- FontAwesome `/fontawesome`, application font (Quicksand) `/fonts`
- Images (logo and favicon) `/images`
- Pictures for testing (split into public and private events) `/pictures/…`

- `Routes`
  - `ROUTES.md` version – VS Code use *MarkDown Preview Enhanced* | PHP Storm native support
  - `routes.yaml` version – VS Code use CTRL +SHIFT + P and select *Preview Swagger* | PHP Storm native supported

---

Task Submission

A SQL dump file (`dump.sql`) must be provided, containing both the schema and any essential seed data along with the entities relationships diagram (ERD) named as `database.png/jpg` showing all tables and relationships present in the live database without cluttering.

Ensure that the module is accessible through URL `ws{XX}.worldskills.uk/module-b` where XX is you station number.

Marking Scheme Summary

SECTION	CRITERIA	TOTAL
B1	Navigation & Routing	1.5
B2	Authentication & Authorisation	3
B3	Public & Private Event Views	2
B4	General layout & Dashboard	2.5
B5	Events Management	3
B6	Pictures Management	3.5
B7	Orders Management	3
B8	Settings	3
B9	Security	1.5
B10	Database	1.5
B11	Maintenance	0.5
	TOTAL ALLOCATED MARKS	**25**