Test Project

Module A: Mini Projects

Web Technologies

Submitted by:

Advaith Jeena, Expert
Pierre Charlier, Expert
Lucas Silveira Portal, Expert
Ling Zhang, Expert
Zoltán Sisák, Expert
Thomas Galih Satria, Expert
Ryoju Ohata, Expert
Seung-Iyul Ryu, Chief Expert
Aizhan Ziyadanova, Expert
Mounir Lahyani, Expert
Merijn Lioen, Expert
Kok Leong Tan, Expert
Chorng-Shiuh Koong, Expert
Valentin Adamescu, Expert
(Thomas) Seng Hin MAK, Independent Test Project Designer

WSC2026_TP17_EN

Contents

Introduction **3**
Description of project and tasks **3**
Starter Task 01: Front-end Skeleton 3
Starter Task 02: Back-end Skeleton 3
Starter Task 03: Back-end with Database 3
Mini Tasks (Follow instructions on which to be implemented) 3
A07-M: Skeleton Loading Cards 3
A12-M: 3D Image Carousel 4
A23-E: Focus Card 5
A25-E: Button 5
B08-E: Favourite Tracks 6
B19-H: Slide CAPTCHA 6
B30-E: Font Hover Effect 7
B40-E: Scroll Ticker 7
C06-M: SQL: March Box Office Revenue 8
C08-M: Job Search & Pagination 8
C09-M: Room Booking Conflict Checker 9
Instructions to the Competitor **9**

© WorldSkills International

Introduction

In this Mini Test Projects, there are two parts, Starter Tasks and Mini Tasks. You are required to implement all starter tasks.

multiple task requirements from different clients. Given the time constraints, you are not required to implement all. Please follow the instructions on which mini tasks need to be implemented.

The tasks following the naming convention A/B/C, Numeric Index, E/M/H for Easy, Medium, Hard, and the name. A tasks are design implementation and browser technologies. B tasks are front-end development. C tasks are back-end development.

Description of project and tasks

Please make sure you implement the following starter task to get started the competition environment. Create one repository for each starter task.

Starter Task 01: Front-end Skeleton

Create a web page that display the current browser time. And the time refreshes every second to display the latest time.

Starter Task 02: Back-end Skeleton

Create a server-side rendered page that display the current server info and server time. It is up to you on what info to display. The only requirement value is the server time.

Starter Task 03: Back-end with Database

In the media asset folder, you will find some data records. Insert the data into the MySQL database and display the data in a table in HTML.

Mini Tasks (Follow instructions on which to be implemented)

Please implement the tasks: A07-M, A12-M, A23-E, A25-E, B8-E, B19-H, B30-E, B40-E, C6-M, C8-M, C9-M.

A07-M: Skeleton Loading Cards

The Morning Wire loads article cards from a slow CMS, showing empty white boxes while they load. The client wants industry-standard **skeleton placeholders with a shimmer sweep** that match the real card layout, so the page feels alive while loading.

The grid has one fully-styled "loaded" card (the reference) and two skeleton cards whose placeholder elements (.sk plus modifier classes) exist in the HTML but are completely unstyled. Add CSS below the marker so that:

Requirements

1. **Grey placeholders** — every .sk element renders as a grey block (base #e2e8f0).
2. **Mirror the card** — placeholders match the loaded card's shapes and positions: image area (~150px tall, square corners), category chip (pill ~84×18px), two large title lines (second ~70% width), two text lines plus one ~40% line — following the modifier classes (sk--image, sk--chip, sk--line, sk--line-lg, sk--w70, sk--w40).

© WorldSkills International

3. **Shimmer sweep** — a light highlight travels left→right across every placeholder and loops infinitely (~1.4s), via a gradient + keyframe animation.
4. **Clip it** — the shimmer stays inside each placeholder (no overflow past the rounded shapes).

Restrictions

- CSS only — no HTML changes, no JavaScript; work below the marker in css/style.css

A12-M: 3D Image Carousel

Write a CSS stylesheet that turns the provided HTML into a continuously rotating, three-dimensional carousel of 12 images arranged on the lateral surface of a vertical cylinder.

Provided

- index.html — the HTML structure. Do not modify it.
- images/1.jpg … images/12.jpg — the 12 images.
- An empty style.css to fill in. This is the only file you must edit.

The HTML already exposes two CSS custom properties for you to consume:

Variable	Where	Meaning
--n	on .a3d	total number of cards (12)
--i	on each .card	the card's zero-based index (0 to 11)

What you must produce

A single style.css that, when linked from the HTML, renders the page as described below.

Visual requirements

Layout

- The 12 cards are placed evenly around the Y axis, forming a regular 12-sided polygon when viewed from above.
- Adjacent cards sit edge-to-edge (with a small constant gap) they must not overlap.
- The carousel is centered both horizontally and vertically in the viewport.
- Anything outside the scene is clipped — no horizontal nor vertical scroll bars.

Cards

- Width: 17.5em.
- Rounded corners (~1.5em).
- The back face of a card must not be visible from behind (the cards on the far side of the cylinder should be hidden, not show through reversed).

3D perspective

- A perspective must be applied so distant cards appear smaller than near ones.
- The cylinder's interior structure (the depth) must be preserved through nested transforms.

Animation

- The whole carousel rotates continuously around its Y axis.
- Constant speed (linear timing), infinite loop, no visible reset.

Constraints

- **No JavaScript.** CSS only.
- **Do not modify the HTML** or rename files.
- **Use the custom properties --n and --i.** The radius and the per-card angle must be computed from --n — do not hardcode 30deg, 12, or the radius as a fixed length anywhere a variable would do.
- Your solution must therefore continue to work if --n and the number of <img> tags are changed together to any other value (e.g. 8, 16, 20).

Hint

The angle between two adjacent cards is 1turn / var(--n). For cards of width w to lie edge-to-edge on a regular n-sided polygon, the cylinder radius is (w/2) / tan(angle/2). CSS calc() supports tan() natively — no preprocessor needed.

A23-E: Focus Card

Implement a card hover effect, using **CSS only**. The basic HTML and CSS styles (three centred image cards) are already provided. Watch example.mp4 for the exact target behaviour.

Requirements

- Three cards sit side by side in .container (already styled: 250×400 px, rounded corners, background photos from assets/). Each card shows a title and subtitle in .card-content.
- **Hovering a card:**
  - the card's text (.card-content) is **revealed with an animation** (it is hidden / offset at rest and slides/fades in on hover);
  - the card's **background image shifts to the right** (e.g. background-position animates towards 100% 0).
- **The other cards** (siblings of the hovered one) are **blurred and scaled down** so the hovered card stands out; the hovered card itself stays sharp at full scale.
- All changes transition **smoothly** and revert when the mouse leaves.

Restrictions

- HTML and CSS only — no JavaScript.
- Do not modify index.html; write all your CSS in style.css.

A25-E: Button

Build and style a button with the hover and press animation shown in example.mp4. Watch the video first — it is the specification for the animation.

Requirements

1. **Button element** — add a <button> labelled **"Button"** to the page, centred horizontally and vertically in the viewport (e.g. flexbox on body), on a light neutral page background (reference: #f0f0f0).
2. **Resting style** — button colour **#725ac1**: #725ac1 bold text (reference: font-size: 25px, padding: 25px 50px) on a light/transparent background, a **2px solid #725ac1 border**, border-radius: 15px, and a pointer cursor.
3. **Hover fill** — on hover the button fills with #725ac1 **from the bottom upwards** and the text turns white. The reference uses an inset box-shadow (box-shadow: inset 0 -100px 0 0 #725ac1); any technique that produces the same bottom-up fill is acceptable.
4. **Press** — while pressed (:active) the button visibly scales down (reference: transform: scale(0.9)).

5. **Motion** — all state changes animate smoothly (reference: transition ≈ 0.3s) and revert cleanly on mouse-out / release.

Restrictions

- HTML and CSS only — no JavaScript.
- Everything in the single index.html; no external libraries, fonts, or assets.

B08-E: Favourite Tracks

Wavelength's "Road Trip Mix" should let listeners mark favourite tracks and see how many they've picked — remembered while the tab stays open. Each li.track has a heart button (.fav); favouriting a track is the .track--fav class on it (already styled). The playlist and favCount elements and the storage key are provided above the marker. Add JS below it to deliver:

Requirements

1. **Toggle favourite** — clicking a track's heart toggles .track--fav on that track (favourite ↔ not).
2. **Live count** — favCount always shows the current number of favourited tracks.
3. **Remember** — save the favourites to sessionStorage on every change and restore them on reload. Because it is *session* storage, the favourites survive a refresh but reset when the tab is closed and reopened.

Restrictions

- Don't change the HTML or CSS; work below the marker in js/script.js

B19-H: Slide CAPTCHA

Build a slide-to-fit image CAPTCHA. A random photo is shown with a small square piece cut out of it; the missing fragment sits at the left edge of the image. The user drags a slider — **horizontally only** — to slide the fragment across until it fills the gap. When it lands close enough, the challenge succeeds and a fresh one loads automatically. Watch the reference video captcha-example.mp4 for the expected behaviour.

The HTML and CSS are fully provided. The scene is **600 × 400 px**, the hole and the piece are **60 × 60 px** squares, and the hole always sits on a **fixed row** (HOLE_Y = 100 px from the top). These constants are already declared in the script.js stub — use them.

Requirements

1. **Random image** — each challenge uses one image chosen at random from the five provided (assets/1.jpg … assets/5.jpg, listed in the IMAGES array). Draw it as the **scene background** (the CSS already sizes backgrounds to 600 × 400).
2. **Random cut-out** — each challenge picks a **random target X** for the hole on the fixed row HOLE_Y. The target must be **at least 100 px from the left edge** (so the puzzle never starts already solved) and the hole must stay fully inside the scene. Show #hole (the white square) at that position.
3. **The piece shows the missing fragment** — render #piece with the same image as its background, offset via background-position so the piece displays **exactly the square that was cut out** at (targetX, HOLE_Y). The piece starts at the **left edge** of the scene (x = 0), on the same row as the hole.
4. **Horizontal-only slider** — dragging #handle with the pointer (use **Pointer Events**: pointerdown / pointermove / pointerup, with pointer capture) moves the piece horizontally, in proportion to the handle's travel, and the handle tracks the piece position. The piece **never moves vertically** and is **clamped** so it stays fully inside the scene at both ends.
5. **Tolerance check** — when the pointer is released, if the piece's X position is within **TOLERANCE (5 px)** of the target: snap the piece exactly onto the target, show the **success state** (add the solved class

to #scene and a success message in #status — the CSS for both is provided) and stop responding to further drags. If it is outside the tolerance, the piece simply stays where it was released and the user can keep dragging.
6. **Refresh on success** — about 1.5 s after a success, a **new challenge loads automatically**: a new random image, a new random target X, the piece and handle reset to the left edge, and the success state cleared.

Restrictions

- Vanilla JS only — no libraries, frameworks or CDNs (offline environment).
- Modify script.js only; index.html and style.css must remain as provided.
- No page reload for the refresh — the new challenge is generated in place.

B30-E: Font Hover Effect

Implement a mouse-responsive text effect using JavaScript.

Implement the following behavior.

- The text "WorldSkills Shanghai" must be displayed in the center of the page.
- When the mouse moves, the thickness of each letter must change based on its distance from the mouse.
- Letters closer to the mouse must appear thicker, and letters farther away must appear thinner.
- All letters must respond simultaneously, each based on its own distance.
- The thickness change must be continuous and smooth, not stepped.

See reference.png for the expected look and demo.mp4 for the expected behavior.

Requirements

1. **Headline** — the text "WorldSkills Shanghai" is displayed in the center of the page.
2. **Live response** — the weight/thickness of the letters changes in real time as the mouse moves (no click needed).
3. **Near = thicker** — letters nearest the cursor become visibly thicker (heavier font weight) than the rest.
4. **Far = thinner** — letters far from the cursor become visibly thinner (lighter weight) than the near ones.
5. **Per-letter independence** — every letter reacts independently and simultaneously, each weight determined by that letter's own distance to the cursor (a smooth gradient of weights across the word), not all letters changing together.
6. **Smooth interpolation** — the thickness change is continuous and smooth (interpolated), not stepped/jumpy between discrete weights.

B40-E: Scroll Ticker

Implement a scroll-driven flowing text using HTML, CSS, and JavaScript.

Implement the following behavior:

- The text "WORLDSKILLS" must be displayed in the center of the page.
- When the mouse wheel is scrolled, new texts must appear from the top of the screen, pass through the center, and disappear toward the bottom, creating a continuous flow.
- Scrolling in the opposite direction must reverse the flow.
- The text closest to the center must appear largest and sharpest, while texts toward the top and bottom must appear tilted and receding with perspective.
- The flow must continue without interruption.

Refer to reference.png for the expected look and demo.mp4 for the expected behavior.

C06-M: SQL: March Box Office Revenue

The Regal Picture House wants the March 2026 box-office league table: how much each movie earned (excluding cancelled bookings), listing only the movies that made meaningful money. The six tables — ratings, movies, screens, customers, screenings, bookings — are shown in the schema.

Requirements

Write **one SELECT** that returns:

1. Columns in this order: movie_title, total_revenue — revenue is SUM(seats × price_per_seat) rounded to 2 decimals.
2. Count only completed bookings (cancelled excluded).
3. Count only screenings in **March 2026** (a February screening must not contribute).
4. Exclude movies whose total revenue is **£500 or less**.
5. Ordered by total_revenue descending, then movie_title ascending for ties (the data has an exact revenue tie).

Restrictions

- Standard SQL only; don't modify the page files.

How to run

Open media/index.html in your browser — no server. The query runs live as you type; when the badge shows **PASS**, click **Save SQL** to write result/result.sql (your submission).

Reference

media/sample/ — the page in a PASS state (the editor query masked).

C08-M: Job Search & Pagination

Northbridge's job board lists 27 vacancies on one endless page. The owner wants the listings endpoint to support a **search box** and **paging**, like every job site. The starter GET /jobs returns every job, unfiltered and unpaginated, from data/jobs.json.

Requirements

Extend **GET /jobs?search=&page=** so it responds 200 with data plus a meta block:

```
{
  "data": [ { "id": 1, "title": "...", "company": "...", "location": "...", "salary": 68000 } ],
  "meta": { "total": 27, "page": 1, "per_page": 10, "total_pages": 3 }
}
```

1. **Search** — when search is given, keep only jobs whose **title or company** contains the term (case-insensitive substring). Empty/absent search = all jobs.
2. **Pagination** — **10 per page**; page defaults to 1.
3. **Meta** — every response includes meta: total (matching count), page, per_page (10), total_pages (ceil(total / 10), minimum 1).
4. **Combined** — search and paging work together (paging applies to the filtered set).
5. **Out of range** — a page past the last returns an empty data array with still-valid meta.

Restrictions

- No frameworks or dependencies — plain PHP, or plain Node http only.

© WorldSkills International

- All responses JSON; keep the starter's 404 for unknown routes.
- Bruno API files are provided.

C09-M: Room Booking Conflict Checker

Harbour House co-working takes meeting-room bookings by phone and keeps double-booking. They want the booking endpoint to validate every request — capacity, opening hours, and clashes with existing bookings. The starter serves GET /rooms (3 rooms with capacities) and GET /bookings (10 existing bookings) from data/.

Requirements

Implement **POST /book** — JSON body { "room_id", "date": "YYYY-MM-DD", "start": "HH:MM", "end": "HH:MM", "attendees" } — validating **in this order**:

1. **Room exists** — unknown room_id → 404.
2. **Capacity** — attendees between 1 and the room's capacity → else 422 "Room capacity exceeded".
3. **Opening hours** — the whole slot within **08:00–20:00** → else 422 "Outside opening hours".
4. **Time range** — end strictly after start → else 422 "Invalid time range".
5. **No overlap** — no clash with an existing booking for the **same room and date**; slots that merely **touch** (one ends exactly as the other starts) are allowed → else 409 "Time slot conflicts with booking <id>".
6. **Success** — 201 with { "data": { booking } }, the new id = highest existing id + 1.

Restrictions

- No frameworks or dependencies — plain PHP, or plain Node http only.
- All responses JSON; keep the starter's GET routes and 404 for unknown routes.
- Bruno API files are provided.

Instructions to the Competitor

- Please create one repository for each task implementation.
- Name your repository with the code of the tasks. For example, Starter01, Starter02, Starter03, A01, B02, C15 etc.
- Do not create multiple repositories for the same task.

© WorldSkills International