S**aved_jobs module** (DB + backend + frontend).

---

1. Database layer (MySQL)

1. **Table already created**
    - Table: saved_jobs

---

2. Backend – API design

2.1 Decide endpoints

Use a small REST-style set:

- GET /api/saved-jobs → list all jobs saved by logged-in user
- POST /api/saved-jobs → save a job
    - Body: { jobId }
- DELETE /api/saved-jobs/:jobId → remove a saved job

(Or a POST /api/saved-jobs/toggle if you prefer one-button toggle.)

2.2 Add controller functions

In your **Node/Express backend**:

1. **Create a controller file**
   e.g. controllers/savedJobsController.js

Implement three functions:

- getSavedJobs(req, res)
    - Read req.user.id from auth middleware.
    - SELECT rj.* FROM saved_jobs sj JOIN recruiter_jobs rj ON sj.job_id = rj.id WHERE sj.user_id = ?
- saveJob(req, res)
    - Read userId from req.user.id, jobId from req.body.jobId.
    - Insert into saved_jobs.
    - Handle duplicate key (already saved) gracefully.
- removeSavedJob(req, res)
    - Read userId from req.user.id, jobId from req.params.jobId.

        - DELETE FROM saved_jobs WHERE user_id = ? AND job_id = ?.

2. **Use existing DB pool**
    - Import your MySQL pool (same way other routes do).
    - Wrap queries with try/catch and return JSON errors (res.status(500).json({message: ...})).
3. **Protect with auth middleware**
    - All saved-jobs APIs must require login (JWT cookie + requireAuth).

2.3 Wire routes

1. In your **routes file** (e.g. routes/router.js / routes/savedJobs.js):
2. const router = require('express').Router();
3. const requireAuth = require('../middleware/requireAuth');
4. const { getSavedJobs, saveJob, removeSavedJob } = require('../controllers/savedJobsController');
5.
6. router.get('/saved-jobs', requireAuth, getSavedJobs);
7. router.post('/saved-jobs', requireAuth, saveJob);
8. router.delete('/saved-jobs/:jobId', requireAuth, removeSavedJob);
9.
10. module.exports = router;
11. In your **main index.js / app.js**, mount:
12. const savedJobsRoutes = require('./routes/savedJobs');
13. app.use('/api', savedJobsRoutes);
14. **If you update logic later**
    - Keep the same URL contract if possible.
    - If you change response shape, update frontend usage accordingly.

---

3. Frontend – UI & API calls

You already have:

- api axios instance with baseURL & withCredentials

- Dashboard links like /dashboard/saved in the footer

3.1 Add Saved Jobs page (dashboard)

1. **Create page component**
   e.g. src/modules/user/Saved.jsx

Responsibilities:

   - On mount, call api.get('/saved-jobs').
   - Store results in savedJobs state.
   - Render job cards (title, company, city, job type).
   - Provide “Remove” button for each (calls DELETE).

2. **Connect route**
   - In router.jsx, add:
   - { path: "/dashboard/saved", element: <Saved /> }
   - You already link /dashboard/saved in Footer so it will start working once route + page are ready.

3.2 Add “Save job” button on job cards

1. In the job listing component (e.g. Jobs list / JobDetail):
   - Add a “Save” / bookmark button:
     - On click, call:
     - await api.post('/saved-jobs', { jobId: job.id });
   - Optionally:
     - Keep a boolean isSaved state per job.
     - On successful POST, set isSaved(true) so the button changes to “Saved”.
2. If you support “Unsave” on the card:
   - Use DELETE /saved-jobs/:jobId
   - Toggle isSaved false.
3. **Handle feedback**
   - Show simple messages: “Saved!” / “Removed from saved”.
   - Disable button while request is in-flight.

3.3 Keep UX consistent

- Show “Saved Jobs” entry in any **user dashboard sidebar** if you have one.
- On Saved.jsx:
    - If savedJobs.length === 0, show empty state:
        - “You haven’t saved any jobs yet. Browse jobs and click ‘Save’ to bookmark them.”
