Community Resource Sharing — Complete Guide

Community Resource Sharing — Complete Step-by-Step Guide
==========

This guide walks a student from cloning the repo to running, extending, and testing the Community Resource Sharing project (Coogs R Us).

Sections
1. Quick start (Docker)
2. Local development (frontend + backend)
3. Database seeding & images
4. Implementing image upload (hands-on)
5. Notifications & request lifecycle
6. Tests & CI notes
7. Troubleshooting checklist
8. Exercises and extensions

---

1) Quick start (Docker)

- Prereqs: Docker Desktop with Compose enabled.
- From repo root:

  ```powershell
  git clone <your-repo-url>
  cd community-resource-sharing-platform
  docker-compose up -d --build
  ```

- Visit the running services:
  - Frontend: http://localhost:3000
  - Backend health: http://localhost:5000/

- Notes:
  - If you need to reset the DB (destructive):
    ```powershell
    docker-compose down -v
    docker-compose up -d --build
    ```

2) Local development (fast iteration)

- Frontend (React dev server):
  ```powershell
  ```

```
cd client
npm install
npm start
```
- This runs CRA hot-reload on http://localhost:3000 and talks to the backend at http://localhost:5000

- Backend (Express dev):
```powershell
cd server
npm install
npm run dev
```
- Ensure `.env` contains DB connection info or use the docker-compose DB when running locally: DB_HOST=db etc.

3) Database & seeds

- The initial seeds are in `server/init.sql`. Those run only when Postgres initializes a fresh volume.
- To apply seeds to an existing DB without removing the volume, run the helper inside the server container:
```powershell
docker-compose exec server node scripts/apply_seeds.js
```

- Inspect the DB:
```powershell
docker-compose exec db psql -U postgres -d resources_db -c "SELECT * FROM resources LIMIT 5;"
```

4) Hands-on: Implement image upload (end-to-end)

Goal: allow users to upload images from the CreateResource page without rebuilding the client image.

Backend steps (server):

- Add dependency: `multer`.

- Create route `server/routes/uploads.js`:
  - Use `multer({ dest: 'uploads/' })` and `router.post('/', upload.single('file'), ...)`.
  - Return JSON: `{ filename, url: `/uploads/${filename}` }`.

- Serve uploads statically in `server/index.js`:
```js
import path from 'path'
app.use('/uploads', express.static(path.join(process.cwd(), 'server', 'uploads')))
```

- Persist uploads with docker-compose (add volume):
  ```yaml
  services:
    server:
      volumes:
        - ./server/uploads:/usr/src/app/uploads
  ```

Frontend steps (client):

- Add a file input to `client/src/pages/CreateResource.js`:
  - Keep file state and show a preview with URL.createObjectURL(file).

- On submit:
  1. If file present: POST to `/api/uploads` with FormData (key `file`).
  2. Receive `{ filename }` and include `image_filename` in the body when POSTing to `/api/resources`.

- Example upload call (using existing `API` axios instance):
  ```js
  const fd = new FormData();
  fd.append('file', selectedFile);
  const up = await API.post('/uploads', fd, { headers: { 'Content-Type': 'multipart/form-data' } });
  const image_filename = up.data.filename;
  await API.post('/resources', { title, description, category, image_filename });
  ```

5) Notifications and requests lifecycle

- Flow summary:
  - User A creates a resource (resource.user_id = A).
  - User B sends POST /api/requests with resource_id.
  - Server inserts request (status=pending) and inserts a notification for owner A.
  - Owner approves: PATCH /api/requests/:id { action: 'approve' }.
    - Server: decrement requester's credits, increment owner's credits, set status approved, notify requester.

6) Tests & CI (suggested starter)

Community Resource Sharing — Freshman-Friendly Build Guide
==========

Goal: walk a beginner through building the Community Resource Sharing app incrementally. Each "checkpoint" has a short goal, commands to run, small code edits, and a test you can perform. Commit after every checkpoint.

Prerequisites
- Git and a code editor (VS Code recommended)
- Node.js (v18+), npm

- Docker Desktop (optional if you prefer running with containers)

How to use this guide
- Follow checkpoints in order.
- Make small commits at each checkpoint with meaningful messages.
- If something breaks, use the Troubleshooting section at the end.

Checkpoint 0 — Prepare the repo
Goal: clone the repo and run the app with Docker to see the baseline.

Commands:
```powershell
git clone <your-repo-url>
cd community-resource-sharing-platform
docker-compose up -d --build
```

Verify:
- Open http://localhost:3000 — you should see the frontend.
- Open http://localhost:5000 — you should see a small JSON health response.

Commit:
```powershell
git checkout -b feat/checkpoint-0
git add .
git commit -m "chore: initial repo snapshot (checkpoint 0)"
```

Checkpoint 1 — Run backend locally and read the code
Goal: start the server locally and understand its structure.

Commands:
```powershell
cd server
npm install
npm run dev
```

What to inspect (files):
- `server/index.js` — express app mount points and middleware
- `server/routes/*.js` — endpoints for auth, resources, requests, notifications
- `server/db.js` — Postgres pool setup

Tiny task: add a new health endpoint that returns the server time.

Edit `server/index.js` (add the route):
```js
app.get('/healthz', (req, res) => res.json({ ok: true, time: new Date().toISOString() }));
```

Test:
```powershell
curl http://localhost:5000/healthz
```

Commit:
```powershell
git add server/index.js
git commit -m "feat: add /healthz endpoint (checkpoint 1)"
```

Checkpoint 2 — Run client locally and inspect
Goal: start CRA dev server, find the resource-listing component, and try editing text.

Commands:
```powershell
cd client
npm install
npm start
```

What to inspect:
- `client/src/App.js` — app shell and routing
- `client/src/pages/Home.js` — resource list entry point
- `client/src/components/ItemCard.js` — how a resource card is rendered

Tiny task: change the app title in `client/src/App.js` to your name + " Coogs R Us".

Test: refresh http://localhost:3000 and confirm the title changed.

Commit:
```powershell
git add client/src/App.js
git commit -m "chore: personalize app title (checkpoint 2)"
```

Checkpoint 3 — Understand auth flow (register/login)
Goal: register a test user and confirm token usage.

Steps:
1. Use the UI (Register) or curl to create a user:
```powershell
curl -X POST http://localhost:5000/api/auth/register -H "Content-Type: application/json" -d
'{"name":"Alice","email":"alice@example.com","password":"password"}'
```
2. Login to receive a JWT token:
```powershell
curl -X POST http://localhost:5000/api/auth/login -H "Content-Type: application/json" -d
```

'{"email":"alice@example.com","password":"password"}'
```

3. Inspect `client/src/api.js` to see how the token is attached to requests via Axios interceptors.

Commit:
```powershell
git commit -am "docs: document auth flow (checkpoint 3)"
```

Checkpoint 4 — Create a resource (backend + client) without images
Goal: POST a resource and verify it appears in the frontend list.

Backend:
- The `POST /api/resources` endpoint accepts title, description, category. No file upload yet.

Client:
- The CreateResource page submits the form to `/api/resources`.

Task:
1. Create a resource via curl (replace token):
```powershell
curl -X POST http://localhost:5000/api/resources -H "Content-Type: application/json" -H
"Authorization: Bearer <token>" -d '{"title":"Free Textbook","description":"Intro to
CS","category":"Textbook"}'
```
2. Visit the app and confirm the new item appears.

Commit:
```powershell
git commit -am "feat: create resource (checkpoint 4)"
```

Checkpoint 5 — Add image upload (hands-on)
Goal: implement an upload endpoint and wire the CreateResource form to upload images first, then create the resource using the returned filename.

Backend changes (small, testable):
1. Install multer in `server`:
```powershell
cd server
npm install multer
```
2. Create `server/routes/uploads.js` with this minimal code:
```js
import express from 'express';
import multer from 'multer';
import path from 'path';
```
```

```
const router = express.Router();
const upload = multer({ dest: path.join(process.cwd(), 'server', 'uploads') });

router.post('/', upload.single('file'), (req, res) => {
  if (!req.file) return res.status(400).json({ message: 'no file' });
  res.json({ filename: req.file.filename, url: `/uploads/${req.file.filename}` });
});

export default router;
```
3. In `server/index.js` mount the route and serve static uploads:
```js
import path from 'path';
app.use('/api/uploads', uploadsRoutes);
app.use('/uploads', express.static(path.join(process.cwd(), 'server', 'uploads')));
```

Client changes (small):
1. Update `client/src/pages/CreateResource.js` to include a file input and file state. On submit, upload the file first with FormData. Example:
```js
const fd = new FormData();
fd.append('file', selectedFile);
const up = await API.post('/uploads', fd, { headers: {'Content-Type': 'multipart/form-data'} });
await API.post('/resources', { title, description, category, image_filename: up.data.filename });
```

2. Show a preview using `URL.createObjectURL(selectedFile)`.

Test:
- Upload a small image and verify it is displayed on the resource card.

Commit:
```powershell
git add server/routes/uploads.js client/src/pages/CreateResource.js server/package.json server/index.js
git commit -m "feat: add uploads endpoint and client upload flow (checkpoint 5)"
```

Checkpoint 6 — Persist uploads with Docker
Goal: make uploaded files survive container restarts.

Edit `docker-compose.yml` to mount uploads:
```yaml
services:
  server:
    volumes:
      - ./server/uploads:/usr/src/app/uploads
```

Then rebuild server:
```powershell
docker-compose build server
docker-compose up -d server
```

Test: upload a file, restart the server container, and re-open its `/uploads/<filename>` URL.

Commit:
```powershell
git add docker-compose.yml
git commit -m "chore: persist uploads with docker volume (checkpoint 6)"
```

Checkpoint 7 — Request flow & notifications
Goal: walk through making requests and approving them.

Steps:
1. As a second user, request a resource: `POST /api/requests { resource_id }`.
2. Owner views requests at `/api/requests/owner` and approves with `PATCH /api/requests/:id { action: 'approve' }`.
3. Confirm notifications are created for both owner and requester.

Test: UI shows notifications and unread badge updates.

Commit:
```powershell
git commit -am "feat: request lifecycle docs and test (checkpoint 7)"
```

Checkpoint 8 — Add tests (jest + supertest)
Goal: write a simple server test for register/login and resource creation.

Install test deps:
```powershell
cd server
npm install --save-dev jest supertest
```

Create `server/tests/auth.test.js` with a simple flow (import app or start server programmatically) and assert that register returns 201 and login returns a token.

Run tests:
```powershell
cd server
npm test
```

Commit:
```powershell
git add server/tests
git commit -m "test: add auth tests (checkpoint 8)"
```

Final checklist before submission
- Ensure README.md documents how to run the stack and which ports to use.
- Make sure your git history contains clear checkpoint commits.
- Optional: write a short demo video or screenshots and include in the PR/zip.

Troubleshooting tips
- Blank UI: open browser DevTools → Console and Network to find the first error.
- Server failing to start: check `docker-compose logs --tail=200 server` or run `npm run dev` locally to see stack traces.
- Database seeds not present: the postgres init.sql runs only when the volume is empty. Use `docker-compose down -v` to reset or run the seed helper.

Exercises and extensions (extra credit)
- Add image resizing using `sharp` and serve thumbnails.
- Move uploads to S3 with presigned upload URLs.
- Add a `mark-all-read` notifications endpoint and a client button.
- Add migrations with `node-pg-migrate` and CI integration (GitHub Actions).

If you'd like, I can implement one of the checkpoints as a runnable branch (for example: full upload feature + tests) and open a draft PR for you. Tell me which checkpoint you want automated and I will implement it and run smoke tests.