Class Companion (ProtectPibble)

A collaborative class tracker where groups share one virtual pet. The pet's health reflects group accountability: missed deadlines damage the pet, consistent completion keeps it alive. The app supports two privacy modes: friend-group transparency and instructor-mode anonymity. The MVP focuses on deadline tracking, completion status, shared pet health, and nudges. AI features are optional stretch goals.

---

1) Goals and non-goals

Goals

- Shared group space per class with:
    - tasks (deadlines, lectures, exams)
    - per-student completion tracking
    - shared pet health that updates based on missed deadlines
    - nudges and activity feed
- Two modes:
    - **Friend mode**: show who did what, leaderboard and damage attribution
    - **Instructor mode**: anonymous or aggregate views, no identities revealed to instructors
- Clean demo for multiple users at once.
- Resume-worthy: real auth, role-based access, database persistence, background job, deployable.

Non-goals for MVP

- Verified attendance (no geolocation, no LMS integration)
- Real grade boosts

- Automatic Piazza scraping
- Perfect slide/PDF parsing pipeline

---

2) Core user stories (MVP)

Shared

1. As a user, I can create or join a group for a class (invite link / code).
2. As a user, I can view all upcoming tasks and my completion status.
3. As a user, I can mark a task complete (or incomplete).
4. As a user, I can see the pet's current health and recent "damage events".
5. As a user, I can send a nudge to a group member about a task.
6. As a user, I can see a group activity feed (who completed, who nudged, penalties applied).

Friend mode (MVP)

7. As a user, I can see member progress and who caused damage.
8. As a user, I can see a simple leaderboard (least missed penalties, most completed tasks).

Instructor mode (MVP)

9. As an instructor, I can create a cohort for a class and share join code.
10. As an instructor, I can upload tasks (deadlines) for the whole cohort.
11. As an instructor, I can view aggregate stats and pet health, without student identities.

---

3) High-level architecture

Frontend

- React + TypeScript
- React Query for data fetching and automatic refetch
- Tailwind + component library for speed

Backend

- FastAPI (Python)
- REST API
- Postgres for state
- Background worker for “deadline penalty” processing
- JWT auth verification (Clerk recommended for speed)

Sync strategy

- MVP uses polling: frontend refetches group state every 10–20 seconds (and on actions).
- Stretch: websockets or realtime pubsub later.

---

4) Domain model

Entities

- **User**: authenticated person
- **Class**: e.g., “CPSC 313”

- **Group**: shared space for a class (friend group or instructor cohort)
- **Membership**: links user to group with role
- **Task**: deadline item (assignment, quiz, lecture, midterm)
- **TaskStatus**: per-user completion for a task
- **Pet**: one per group with health/maxHealth
- **Event**: immutable log of changes (missed deadline, completion, nudge)

Roles

- `student`
- `instructor`
- `admin` (optional, internal)

Group modes

- `FRIEND`
- `INSTRUCTOR`

---

5) Database schema (practical MVP)

Use UUIDs for ids. Suggested fields:

users

- id (uuid, pk)
- email (unique)
- display_name

- created_at

classes

- id (uuid, pk)
- school (text)
- code (text) // “CPSC 313”
- term (text) // “2026W”
- created_at

groups

- id (uuid, pk)
- class_id (fk)
- mode (enum FRIEND/INSTRUCTOR)
- name (text) // “Kabir’s CPSC313”
- invite_code (unique, short text)
- created_by (user_id)
- created_at

group_memberships

- group_id (fk)
- user_id (fk)
- role (enum student/instructor)
- joined_at
  Primary key: (group_id, user_id)

tasks

- id (uuid, pk)
- group_id (fk)
- title (text)
- type (enum ASSIGNMENT/QUIZ/LECTURE/EXAM/OTHER)
- due_at (timestamp)
- penalty (int) // damage amount
- created_by (user_id)
- penalty_applied_at (timestamp, nullable) // indicates system applied penalties for this task
- created_at

task_status

- task_id (fk)
- user_id (fk)
- status (enum NOT_DONE/DONE/EXCUSED)
- completed_at (timestamp, nullable)
  Primary key: (task_id, user_id)

pets

- group_id (pk, fk)
- name (text)
- health (int)
- max_health (int)
- avatar_url (text, nullable)
- updated_at

events

- id (uuid, pk)
- group_id (fk)
- type (enum TASK_CREATED/TASK_COMPLETED/TASK_MISSED/PET_DAMAGED/NUDGE_SENT)
- actor_user_id (uuid, nullable) // who triggered it (null for system)
- target_user_id (uuid, nullable) // who got nudged or who missed
- task_id (uuid, nullable)
- delta (int, nullable) // health change
- message (text, nullable)
- created_at

Constraints to prevent double-penalties:

- Add unique index for missed event: (type, task_id, target_user_id) where type = TASK_MISSED

---

6) Pet health rules (MVP)

Damage application

- When a task passes its due_at, any member who is NOT marked DONE or EXCUSED receives a penalty.
- Each missed user causes:
    - a TASK_MISSED event
    - pet.health decreases by task.penalty

Floor/ceiling

- pet.health is clamped: 0 ≤ health ≤ max_health

Healing (optional MVP)

Skip healing for hackathon unless you want it.
If you include it, keep it simple:

- completing tasks before due date can add +1 health up to max, or award “streak points” instead of healing.

---

7) API specification (FastAPI)

All endpoints require auth except health check.

Auth model

- Frontend gets JWT from auth provider.
- Backend verifies JWT and extracts user identity.
- Backend creates user row if first login.

Headers

- Authorization: Bearer <JWT>

Group state endpoint (important)

GET /groups/{group_id}/state

Returns everything frontend needs to render the dashboard.

Response (friend mode example):

```
{
  "group": {
    "id": "uuid",
```

```
    "name": "CPSC 313 Squad",
    "mode": "FRIEND",
    "class": { "code": "CPSC 313", "term": "2026W" }
  },
  "pet": {
    "name": "Byte",
    "health": 72,
    "maxHealth": 100,
    "avatarUrl": "https://..."
  },
  "members": [
    { "userId": "uuid", "displayName": "Kabir", "role": "student" },
    { "userId": "uuid", "displayName": "A", "role": "student" }
  ],
  "tasks": [
    {
      "id": "uuid",
      "title": "Quiz 0",
      "type": "QUIZ",
      "dueAt": "2026-01-18T01:00:00Z",
      "penalty": 5,
      "myStatus": "DONE",
      "stats": { "doneCount": 3, "totalCount": 5 }
    }
  ],
  "leaderboard": [
    { "userId": "uuid", "displayName": "Kabir", "missedCount": 0,
"damage": 0, "doneCount": 6 }
  ],
  "recentEvents": [
    {
      "type": "TASK_MISSED",
      "targetUserId": "uuid",
      "taskId": "uuid",
      "delta": -5,
      "createdAt": "2026-01-17T20:00:00Z"
    }
  ]
}
```

Instructor mode differences:

- `members` can be omitted or anonymized.
- `leaderboard` becomes aggregate stats only, no names.

Groups

POST /groups

Create group.
Body:

```
{ "classCode": "CPSC 313", "term": "2026W", "mode": "FRIEND",
"name": "CPSC313 Squad" }
```

POST /groups/join

Join via invite code.
Body:

```
{ "inviteCode": "AB12CD" }
```

GET /groups/my

List groups current user is a member of.

Tasks

POST /groups/{group_id}/tasks

Create a task.
Body:

```
{ "title": "Assignment 1", "type": "ASSIGNMENT", "dueAt":
"2026-01-25T07:00:00Z", "penalty": 10 }
```

PATCH /tasks/{task_id}

Edit task (creator or instructor only).

DELETE /tasks/{task_id}

Delete task (creator or instructor only).

POST /tasks/{task_id}/complete

Mark current user as DONE.
Body:

```
{ "status": "DONE" }
```

Also allow EXCUSED for instructor mode:

```
{ "status": "EXCUSED" }
```

Nudges

POST /groups/{group_id}/nudges

Body:

```
{ "toUserId": "uuid", "taskId": "uuid", "message": "Quiz due tonight, lock in" }
```

Creates NUDGE_SENT event.

Events

GET /groups/{group_id}/events?limit=50

Returns recent events for feed (useful if you want separate feed view).

---

8) Frontend spec (React)

Pages

1. Auth / login
2. Group selector (my groups)
3. Group dashboard
4. Task creation modal (role-based)
5. Optional: group settings page (mode, punishment text)

Dashboard components

- PetCard: health bar, avatar, “alive/dead” state
- UpcomingTasksList: tasks sorted by due date, status chips
- Calendar view (optional, but looks good)
- MemberProgress (friend mode only)
- ActivityFeed: recent events
- NudgeModal: select user + task + message
- Leaderboard (friend mode)

Data fetching rules (important)

Use React Query:

- Query: `groupState`
    - key: `["groupState", groupId]`
    - fetch: `GET /groups/{id}/state`
    - refetchInterval: 15000 ms (15s)
    - also refetch on:
        - mark complete
        - create/edit/delete task

        - send nudge

This is your “sync”.

---

9) Background job spec (deadline penalties)

You need a worker that periodically applies penalties after due dates.

Job: apply_deadline_penalties

Runs every 1–5 minutes.

Algorithm:

1. Find tasks where:
    - due_at < now
    - penalty_applied_at IS NULL
2. For each task:
    - fetch all active group members (students)
    - for each user:
        - if task_status is not DONE and not EXCUSED:
            - insert TASK_MISSED event (idempotent via unique constraint)
            - decrement pet.health by penalty
3. Set task.penalty_applied_at = now
4. Commit transaction

Implementation choices:

- Hackathon-simple: APScheduler in backend (single instance)

- Better: Celery worker + Redis

For resume, Celery looks better, but APScheduler is faster to ship.

---

10) AI feature spec (stretch)

Keep it demo-friendly. Don't promise full slide parsing.

AI pet generation (easy win)

- Prompt based on class code + group vibe + member names
- Generate a pet name + description
- Optional: generate an image avatar
  Store avatar_url in pets table.

AI lecture summary (safe stretch)

MVP approach:

- user pastes lecture text or uploads PDF later
- backend calls model to summarize into bullet points
- store summary attached to a task of type LECTURE or in a "lecture_notes" table

Piazza Q&A (not MVP)

Only do if time remains and Piazza access is easy.
Otherwise phrase as "future integration".

---

11) Privacy and safety rules (important)

Instructor mode constraints

- Instructors can view:
    - pet health
    - total missed counts
    - completion rates per task (doneCount/totalCount)
- Instructors cannot view:
    - which specific student missed (no names, no targetUserId in responses)

Enforce this at the API layer by filtering fields.

---

12) Deployment plan (demo-ready for multiple people)

Services

- Frontend: Vercel
- Backend: Render (FastAPI web service)
- Worker: Render background worker (Celery or scheduler process)
- Postgres: Neon or Supabase Postgres
- Redis: Upstash (only if using Celery/RQ)

Deployment steps

1. Create Postgres instance, set DATABASE_URL in backend.
2. Deploy backend:
    - FastAPI app
    - run migrations (Alembic) on deploy

3. Deploy worker:
    - runs apply_deadline_penalties loop/schedule
4. Deploy frontend:
    - set API_BASE_URL env var
    - set auth provider env vars
5. Demo flow:
    - Person A creates group, shares invite code
    - Person B joins on their laptop
    - Add a task due in 2 minutes with penalty
    - Person A marks done, Person B does nothing
    - Wait for worker to apply penalties
    - Both dashboards update (polling) showing pet health drop and friend-mode attribution

This is a clean multi-user demo story.

---

13) MVP build checklist (what to implement first)

Day 1 (core)

- Auth + user creation
- Create/join group with invite code
- Create tasks
- Mark tasks complete
- GET /groups/{id}/state endpoint

- Pet shown with health bar
- Polling refetch in frontend

Day 2 (polish + wow)

- Background penalty worker
- Activity feed events
- Nudges
- Friend vs instructor mode filtering
- UI polish + leaderboard
- Demo script + seeded sample class

Stretch if time:

- AI pet name/description
- AI summary input box

---

0) One-time team decisions (do immediately)

- Use a **monorepo** with `frontend/` and `backend/`
- Use **Docker Compose** for local Postgres (and Redis later if you want)
- Pick a branch strategy: `main` (protected) + `dev` + feature branches

---

1) Create the GitHub repo

1. One person creates a GitHub repo: `class-companion` (private or public).
2. Add teammates as collaborators.
3. Add protections (recommended):
    - Protect `main` (require PR, require 1 review)
    - Use `dev` as the merge branch during hackathon

Clone it:

```
git clone <repo-url>
cd class-companion
```

---

2) Install what you need (Mac)

Everyone should install these:

Required

- **Git**
- **Node.js 20+**
    - easiest: install via Homebrew or nvm
- **Python 3.11+**

- **Docker Desktop** (for Postgres locally)

Strongly recommended

- **Cursor** (you already use it)
- **Postman** or **Insomnia** (API testing)

Quick checks:

```
node -v
npm -v
python3 --version
docker --version
git --version
```

If any of these fail, fix before moving on.

---

3) Repo file structure (monorepo)

Create this structure:

```
class-companion/
  README.md
  .gitignore
  docker-compose.yml
  .env.example

  frontend/
    package.json
    src/
      app/
      components/
      pages/
      lib/
      styles/
    vite.config.ts
    tsconfig.json

  backend/
```

```
pyproject.toml
app/
  main.py
  core/
    config.py
    security.py
  db/
    session.py
    models/
    migrations/
  api/
    routes/
  services/
  schemas/
  workers/
alembic.ini
```

Notes:

- Frontend: I recommend **Vite + React + TS** for speed.
- Backend: **FastAPI** with clean separation.
- `workers/` will later run the penalty scheduler.

---

4) Add Docker Compose for local Postgres

Create `docker-compose.yml` in repo root:

```
services:
  db:
    image: postgres:16
    container_name: class_companion_db
    environment:
      POSTGRES_USER: app
      POSTGRES_PASSWORD: app
      POSTGRES_DB: class_companion
    ports:
      - "5432:5432"
```

```
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Start it:

```
docker compose up -d
```

Test Postgres is running:

```
docker ps
```

---

5) Backend setup (FastAPI)

From repo root:

```
cd backend
python3 -m venv .venv
source .venv/bin/activate
pip install fastapi uvicorn sqlalchemy alembic psycopg2-binary
pydantic-settings python-dotenv
```

Create `backend/app/main.py`:

```
from fastapi import FastAPI

app = FastAPI(title="Class Companion API")

@app.get("/health")
def health():
    return {"status": "ok"}
```

Run backend:

```
uvicorn app.main:app --reload --port 8000
```

Open:

- `http://localhost:8000/health`
- `http://localhost:8000/docs`

---

6) Frontend setup (React + TS)

From repo root:

```
cd frontend
npm create vite@latest . -- --template react-ts
npm install
npm run dev
```

Frontend runs at:

- `http://localhost:5173`

---

7) Environment variables

In repo root, create `.env.example`:

```
DATABASE_URL=postgresql+psycopg2://app:app@localhost:5432/class_companion
API_BASE_URL=http://localhost:8000
```

Each dev creates their own `.env` (not committed) by copying:

```
cp .env.example .env
```

Backend should read env vars (later). For now, health endpoint doesn't need it.

---

8) Add basic scripts to make life easy

In root `README.md`, include "how to run locally":

- Start DB: `docker compose up -d`
- Run backend: `cd backend && source .venv/bin/activate && uvicorn app.main:app --reload --port 8000`
- Run frontend: `cd frontend && npm run dev`