Fitness Tracker

An application that tracks workouts and exercises via a NESTJS API with a React UI

by Sai Thirakala

Project Overview and Goals

Purpose: To build a minimal fitness tracker to create, view, update, and delete workouts and entries within them, using a NestJS API, PostgreSQL database, and a React UI

Goals:

- Create a workout and see it in list immediately
- Add entries to workout and see them render w/o reload
- Edit/delete workouts and entries
- Call all endpoints via a .http file
- Guard PUT/POST/PATCH to require JSON
- Validate inputs with DTOs + ValidationPipe

In Scope:

- Entities: Workout and Entry
- Relation: Workout 1-N Entries
- REST CRUD for both entities
- Frontend: React forms/lists with fetch + async/await
- DB: Postgres (using Docker), IDs and UUIDs
- CORS enable, simple JSON-only Guard on write

Out of Scope:

- User accounts/authentication
- Analytics/charts
- File upload
- Advanced filtering

Concepts Demonstrated: NestJS controllers/services (DI), guards, pipes, DTO validation, TypeORM relations, React components, props, types event handlers, state, list keys, conditional rendering, async fetch

Technology Stack

Frontend	React + TypeScript (Vite)	Dev server on 5173, fetch + async/await, function components, useState
Backend	NestJS, TypeORM	REST controllers/services, DTOs + class-validator, CORS enabled, listens on 3000
Database	PostgresSQL (Docker)	Image: postgres:17, UUID ids, 1-N relation (Workout -> Entry)
Tools	Docker Compose, VS Code REST Client, ESLint/Prettier	DB in Docker, test.http file for endpoint testing

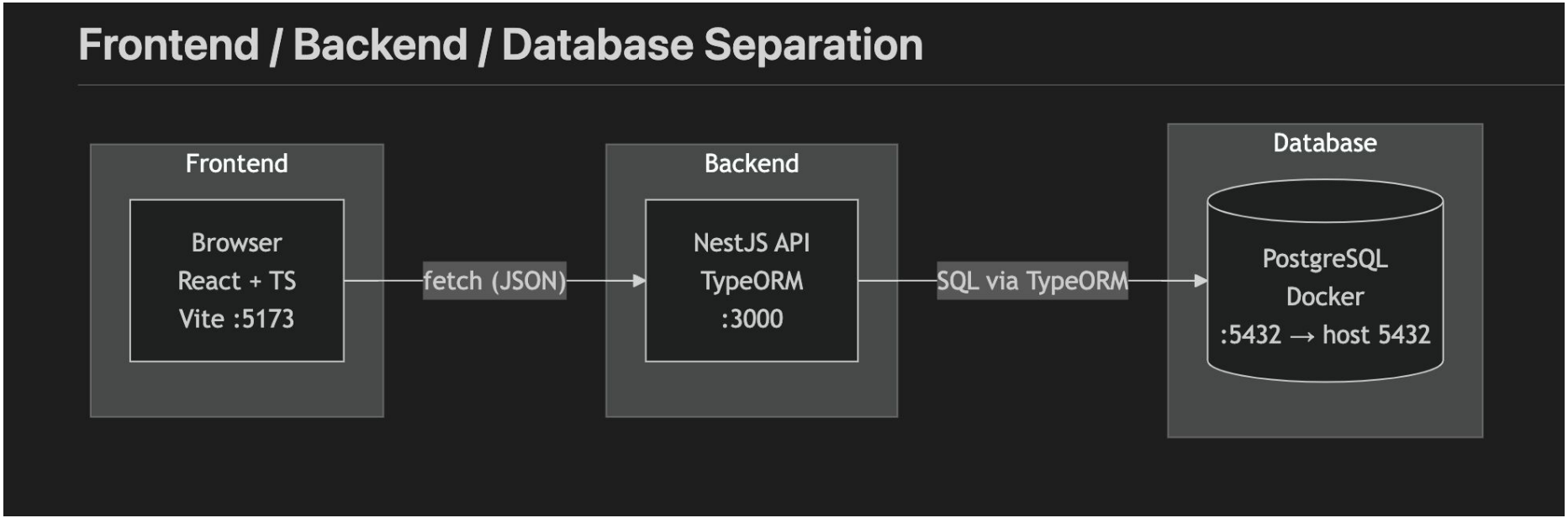
Ports:

- 5173: Vite
- 3000: Nest API
- 5432: Postgres container

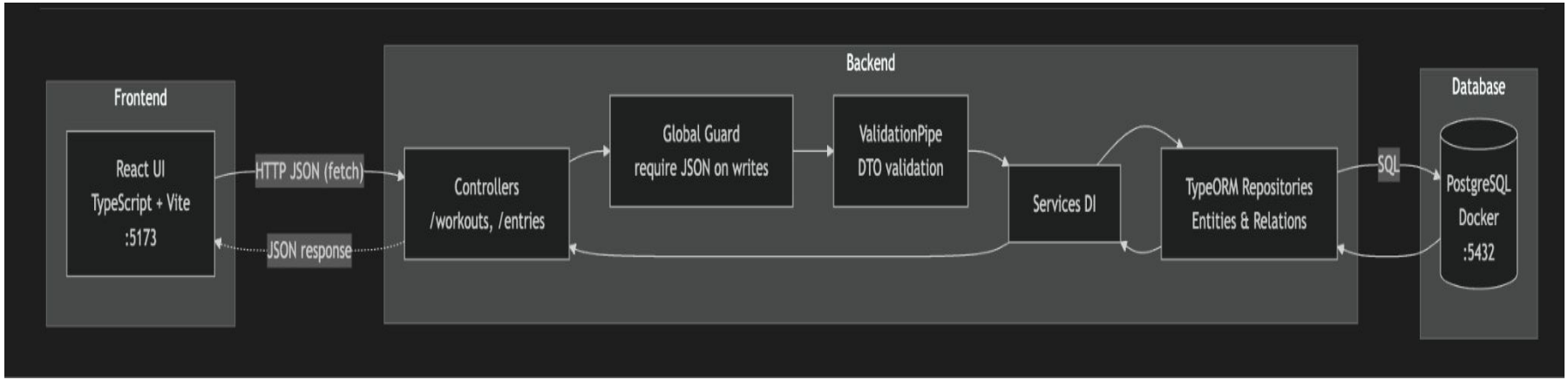
Key Libraries:

- Backend: @nestjs/common, @nestjs/core, @nestjs/typeorm, pg, class-validator, class-transformer, dotenv
- Frontend: react, react-dom
- Dev/Testing: VSCode REST Client, ESLint, TS config

Technology Stack Cont.

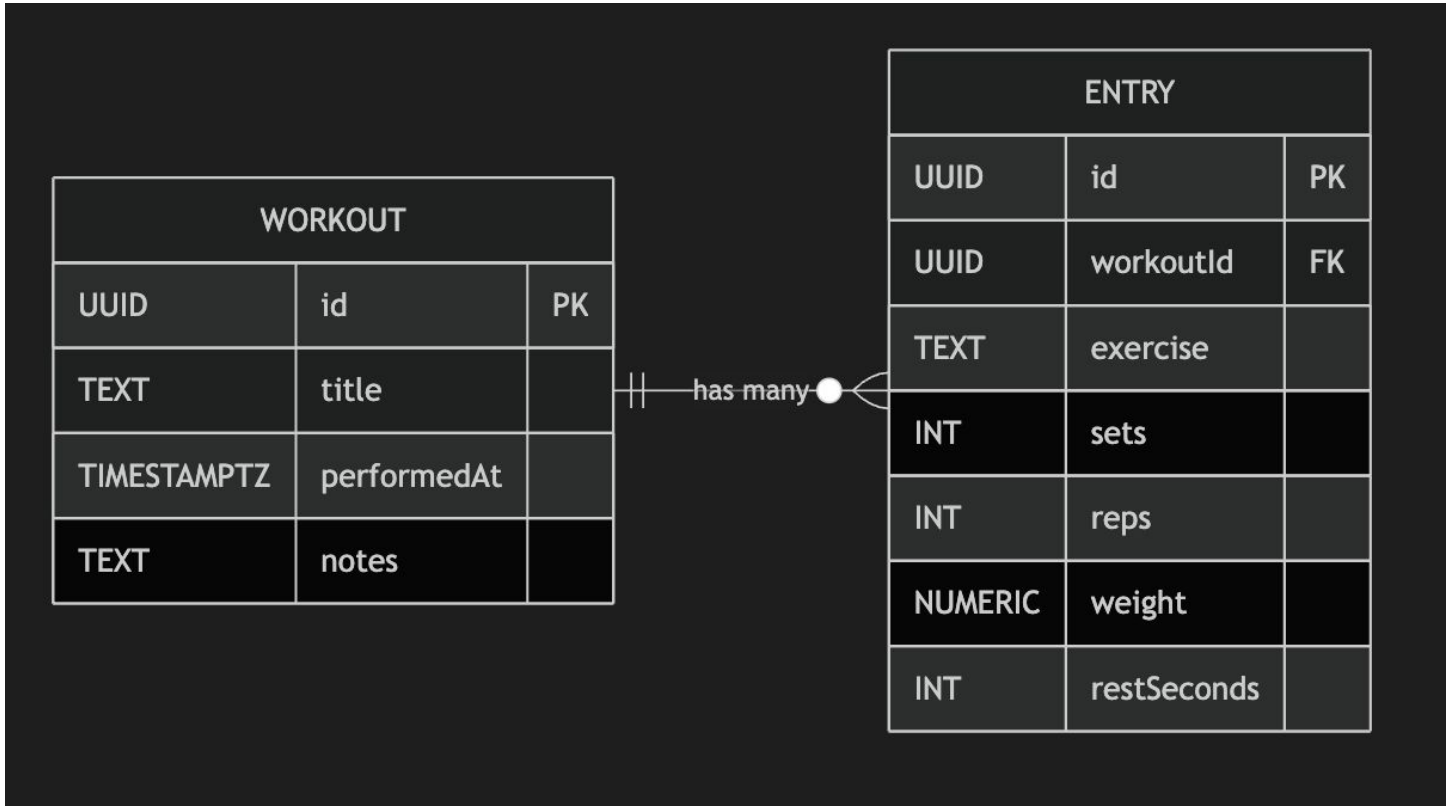


System Architecture Diagram



React UI (Vite 5173) sends a JSON request to the NestJS API (3000). A global guard requires the content type to be application/json on writes. The DTOs are validated using ValidationPipe. TypeORM persists data to PostgreSQL (Docker 5432) and the API returns JSON responses to the UI.

Database Schema and Entities



REST API Design and Endpoints

Method	Route	Desc.	Body	Success	CRUD
GET	/workouts	List all workouts	-	200	Read
GET	/workout/:id	Get one workout + its entries	-	200	Read
POST	/workouts	Create a workout	{ “title”: string, “performedAt”: “ISO-8601”, “notes”?: string }	201	Create
PATCH	/workouts/:id	Update a workout	Any subset of post fields	200	Update
DELETE	/workouts/:id	Delete a workout	-	204	Delete
GET	/entries	List all entries	-	200	Read
GET	/entries/:id	Get one entry	-	200	Read
POST	/entries	Create an entry	{ “workouts”: “uuid”, “exercise”: string, “reps”: number, “weight”?: number, “restSeconds”?: number }	201	Create
PATCH	/entries/:id	Update an entry	Any subset of post fields	200	Update
DELETE	/entries/:id	Delete an entry	-	204	Delete

CRUD Implementation Summary

Op	Resource	Method/Route	Controller Method	Service Method	TypeORM Call	Notes
Create	Workout	POST /workouts	WorkoutsController.create	WorkoutsService.create(dto)	repo.create -> repo.save	Guard requires JSON; DTO validated; returns 201 + Workout/Entry
	Entry	POST /entries	EntriesController.create	EntriesService.create(dto)		
Read (list)	Workout	GET /workouts	WorkoutController.findAll	WorkoutsService.findAll()	repo.find	Returns 201 + Workout[]/Entry[]
	Entry	GET /entries	EntriesController.findAll	EntriesService.findAll()		
Read (one)	Workout	GET /workouts/:id	WorkoutsController.findOne	WorkoutsService.findOne(id, dto)	repo.findOne	404 if not found; returns 200 + Workout/Entry
	Entry	GET /entries/:id	EntriesController.findOne	EntriesService.findOne(id, dto)		
Update	Workout	PATCH /workouts/:id	WorkoutsController.update	WorkoutsService.update(id, dto)	repo.findOne -> repo.save	Guard requires JSON; DTO validates; returns 200 + Workout/Entry
	Entry	PATCH /entries/:id	EntriesController.update	EntriesService.update(id, dto)		
Delete	Workout	DELETE /workouts/:id	WorkoutsController.remove	WorkoutsService.remove(id)	repo.delete	204 on success; 404 if not found
	Entry	DELETE /entries/:id	EntriesController.remove	EntriesService.remove(id)		

CRUD Implementation Summary Cont.

Entries PATCH Snippet:

```
@Patch(':id')
update(@Param('id') id: string, @Body() dto: UpdateEntryDto) {
  return this.service.update(id, dto);
}
```

```
async update(id: string, dto: UpdateEntryDto) {
  const e = await this.findOne(id);
  if (dto.workoutId) {
    const w = await this.workouts.findOne({ where: { id: dto.workoutId } });
    if (!w) throw new BadRequestException('workoutId does not exist');
  }
  Object.assign(e, dto);
  return this.entries.save(e);
}
```

Workouts READ Snippet:

```
@Get(':id')
findOne(@Param('id') id: string) {
  return this.service.findOne(id);
}
```

```
async findOne(id: string) {
  const w = await this.repo.findOne({
    where: { id },
    relations: { entries: true },
  });
  if (!w) throw new NotFoundException('Workout not found');
  return w;
}
```

NestJS Features Used

Feature	Purpose	Where
Controllers	Define routes and shape resources	workouts.controller.ts, entries.controller.ts
Services (DI)	Business logic, injected into controllers	workouts.services.ts, entries.services.ts
Modules	Group related controllers/providers	workouts.module.ts, entries.module.ts, app.module.ts
Guard (global)	Requires JSON on POST/PATCH, 415 is missing	require-json.guard.ts, registered in app.module.ts
Pipes (global)	DTO validation and transformation, 400 on invalid input	maint.ts (ValidationPipe)
DTOs + class-validator	Request body contracts	create-entry.dto.ts, update-entry.dto.ts, create-workout.dto.ts, update-workout.dto.ts
TypeORM	Entities, relation, repos	workout.entity.ts, entries.entity.ts
Config + CORS	Env config + cross-origin requests	app.module.ts, main.ts (enableCors)

NestJS Features Snippets

Global Guard registration

```
@Module({
  imports: [
    ConfigModule.forRoot({ isGlobal: true }),
    TypeOrmModule.forRootAsync({
      useFactory: () => ({
        type: 'postgres',
        url: process.env.DATABASE_URL,
        autoLoadEntities: true,
        synchronize: true,
      }),
    }),
    WorkoutsModule,
    EntriesModule,
  ],
  providers: [{ provide: APP_GUARD, useClass: RequireJsonGuard }],
})
export class AppModule {}
```

Global ValidationPipe + CORS

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(new ValidationPipe({ whitelist: true, transform: true }));
  app.enableCors();
  await app.listen(3000);
}
bootstrap();
```

DTO with validation

```
export class CreateWorkoutDto {
  @IsISO8601() performedAt: string;
  @IsString() @Length(2, 80) title: string;
  @IsOptional() @IsString() @Length(0, 400) notes?: string;
}
```

Dependency Injection in Controller

```
@Controller('workouts')
export class WorkoutsController {
  constructor(private readonly service: WorkoutsService) {}

  @Get()
  findAll() {
    return this.service.findAll();
  }
```

Service + Repo

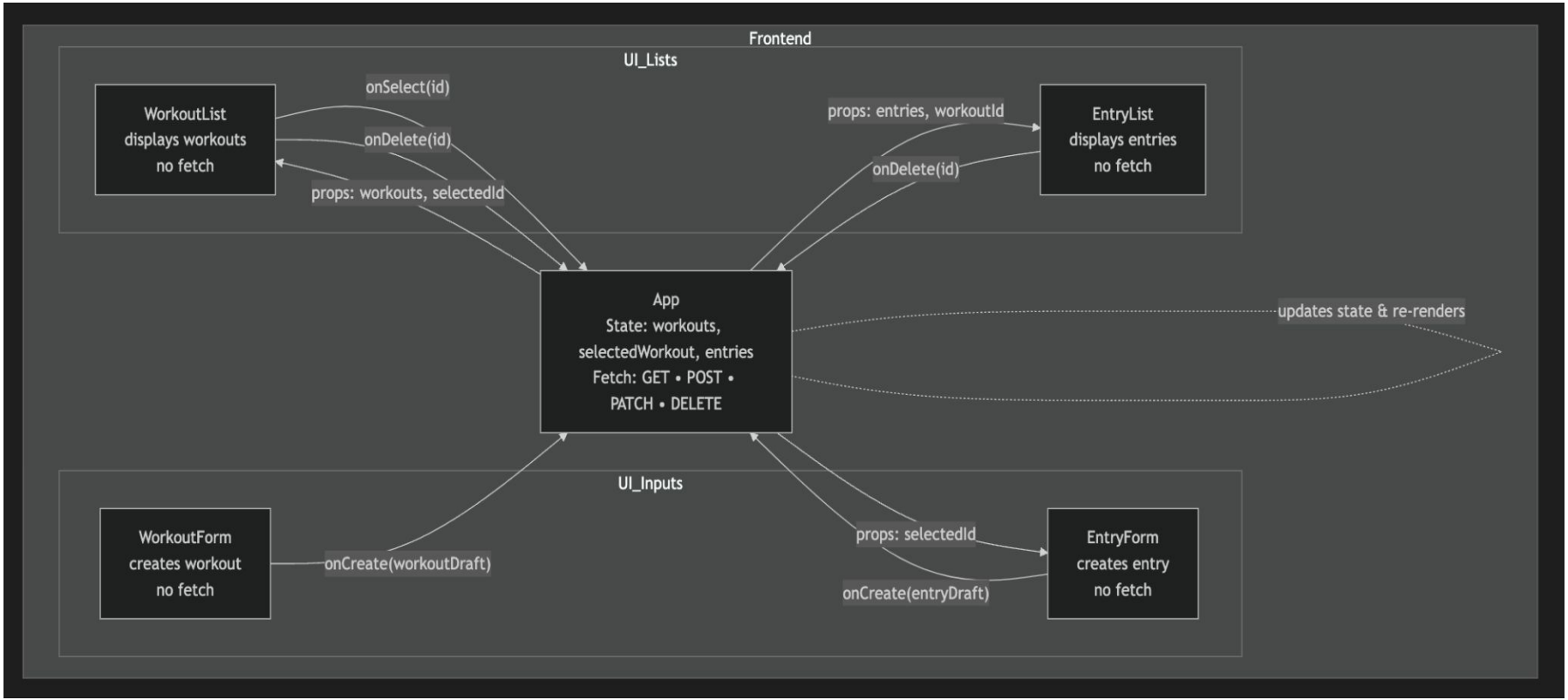
```
@Injectable()
export class WorkoutsService {
  constructor(@InjectRepository(Workout) private repo: Repository<Workout>) {}

  findAll() {
    return this.repo.find({ order: { performedAt: 'DESC' } });
  }

  async findOne(id: string) {
    const w = await this.repo.findOne({
      where: { id },
      relations: { entries: true },
    });
    if (!w) throw new NotFoundException('Workout not found');
    return w;
  }
```

Frontend Overview

Component	Role	Fetches	Props In	Emits
App	State + orchestration; holds workouts, selectedWorkout, entries	Yes	-	-
WorkoutForm	Forms to create a workout	No, delegated to App	-	onCreate(workoutDraft)
WorkoutList	Displays workouts list; select or delete	No, display only	workouts, selectedId	onSelect(id), onDelete(id)
EntryForm	Form to add an entry for selected workout	No, delegated to App	workoutId	onCreate(entryDraft)
EntryList	Displays entries for selected workout; delete	No, display only	entries, workoutId	onDelete(id)

- App performs all fetch calls and updates state
- Forms and lists are presentational + event emitters (they never fetch directly)
- On create/delete/update, App refreshed the relevant lists and passes updates props down

Frontend Overview Cont.



Frontend Features

App (state, fetch, pass props)

```tsx
export default function App() {
  const [selectedId, setSelectedId] = useState<string | undefined>()
  const [detail, setDetail] = useState<WorkoutWithEntries | null>(null)
  const [refreshKey, setRefreshKey] = useState(0)
  const reloadList = () => setRefreshKey(k => k + 1)

  const loadDetail = useCallback(async (id: string) => {
    const res = await fetch(`${API}/workouts/${id}`)
    const w = await res.json() as WorkoutWithEntries
    setDetail(w)
  }, [])

  useEffect(() => { if (selectedId) loadDetail(selectedId).catch(() => setDetail(null)) }, [selectedId, loadDetail])

  function onWorkoutCreated(w: Workout) { setSelectedId(w.id); reloadList() }
  function onWorkoutDeleted(id: string) { if (detail?.id === id) setDetail(null); if (selectedId === id) setSelectedId(undefined) }
  function onEntryCreated(e: Entry) { setDetail(d => d ? { ...d, entries: [e, ...d.entries] } : d) }
  function onEntryDeleted(id: string) { setDetail(d => d ? { ...d, entries: d.entries.filter(e => e.id !== id) } : d) }

  const header = useMemo(() => detail ? `${detail.title} — ${new Date(detail.performedAt).toLocaleString()}` : 'Select a workout', [detail])

  return (
    <div className="container">
      <header>
        <h1>Fitness Tracker</h1>
        <div className="subtle">API: {API}</div>
      </header>

      <main className="grid">
        <section>
          <WorkoutForm onCreated={onWorkoutCreated} />
          <WorkoutList selectedId={selectedId} onSelect={setSelectedId} onDeleted={onWorkoutDeleted} refreshKey={refreshKey} />
        </section>

        <section>
          <div className="card">
            <h2>{header}</h2>
            {!detail && <p>Pick a workout on the left.</p>}
            {detail && (
              <>
                <EntryForm workoutId={detail.id} onCreated={onEntryCreated} />
                <EntryList entries={detail.entries ?? []} onDeleted={onEntryDeleted} />
              </>
            )}
          </div>
        </section>
      </main>
    </div>
  )
```

WorkoutForm (typed props + submit event)

```tsx
export default function WorkoutForm({ onCreated }: Props) {
  const [title, setTitle] = useState('')
  const [performedAt, setPerformedAt] = useState('')
  const [notes, setNotes] = useState('')
  const [busy, setBusy] = useState(false)
  const [err, setErr] = useState<string | null>(null)

  async function submit(e: React.FormEvent) {…
  }

  return (
    <form onSubmit={submit} className="card form">
      <h2>New Workout</h2>
      <label>Title<input value={title} onChange={e=>setTitle(e.target.value)} /></label>
      <label>When<input type="datetime-local" value={performedAt} onChange={e=>setPerformedAt(e.target.v
      <label>Notes<textarea rows={2} value={notes} onChange={e=>setNotes(e.target.value)} /></label>
      {err && <p className="error">{err}</p>}
      <button disabled={busy}>{busy ? 'Saving…' : 'Add Workout'}</button>
    </form>
  )
}
```

WorkoutList (props in, events out)

```tsx
type Props = {
  selectedId?: string
  onSelect: (id: string) => void
  refreshKey?: number
  onDeleted?: (id: string) => void
}

  return (
    <div className="card">
      <h2>Workouts</h2>
      {data.length === 0 && <p>No workouts yet.</p>}
      <ul className="list">
        {data.map(w => (
          <li key={w.id} className={w.id === selectedId ? 'selected' : ''}>
            <button className="link" onClick={() => onSelect(w.id)}>
              <div className="title">{w.title}</div>
              <div className="sub">{new Date(w.performedAt).toLocaleString()}</div>
            </button>
            <button className="danger" onClick={() => del(w.id)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  )
```

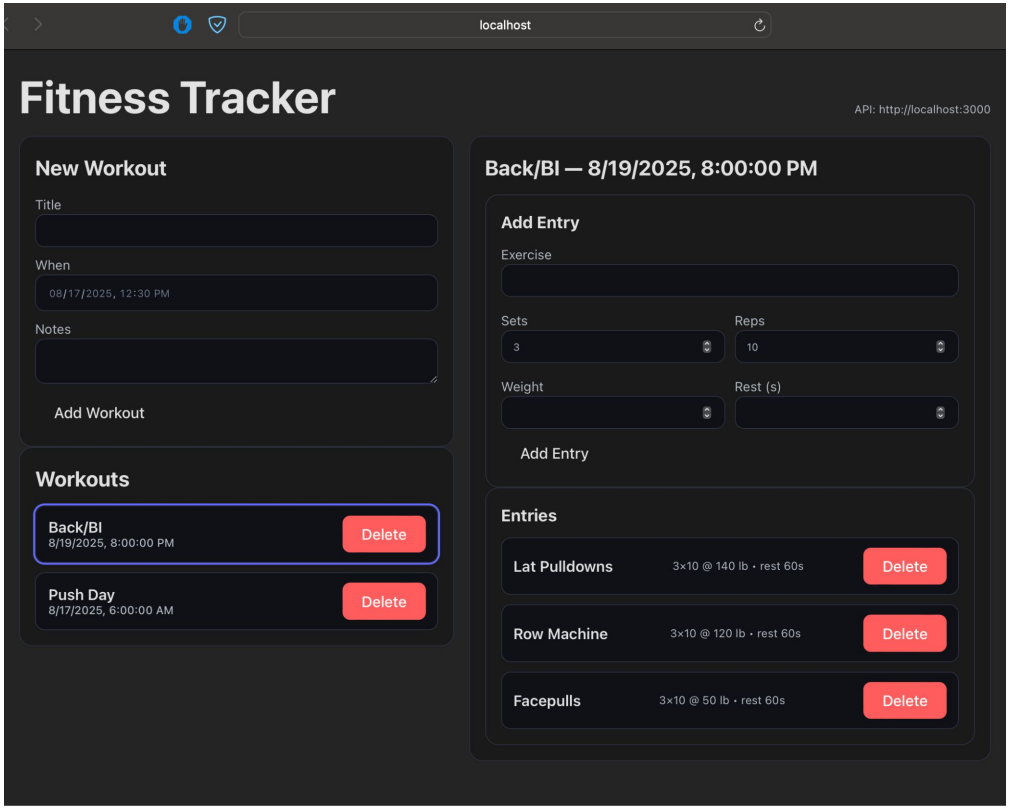
CSS Styling Approach

Method:

- Vanilla CSS in frontend/src/index.css
- Uses CSS variables, system fonts, and prefers-color-scheme for light/dark mode
- Focus-visible outlines + minimal spacing utilities

What I Styled:

- Page layout (centered column)
- Forms (inputs, labels, buttons)
- Lists (workouts, entries) with hover/focus states
- Utility classes: .stack, .row, .card, .muted



Testing with REST Client

POST Request/Response

```
###
# Create another workout to test listing
Send Request
POST {{baseUrl}}/workouts
Content-Type: application/json

{
  "title": "Pull Day",
  "performedAt": "2025-08-18T10:00:00.000Z",
  "notes": "Back/Biceps"
}
```

```
HTTP/1.1 201 Created
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 127
ETag: W/"7f-O8B+ZK7dGH74fO83Iz5bfQvYHPc"
Date: Mon, 18 Aug 2025 00:48:28 GMT
Connection: close

{
  "id": "5087516c-a98a-40d5-9e5c-b2d1cd6fbe6c",
  "performedAt": "2025-08-18T10:00:00.000Z",
  "title": "Pull Day",
  "notes": "Back/Biceps"
}
```

PATCH Request/Response

```
###
# Update workout
Send Request
PATCH {{baseUrl}}/workouts/{{workoutId}}
Content-Type: application/json

{
  "title": "Push Day (Updated)",
  "notes": "Incline focus"
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 152
ETag: W/"98-Uwcb3T5ClVEEQGqpVfvZZLr/Pb8"
Date: Mon, 18 Aug 2025 00:51:00 GMT
Connection: close

{
  "id": "58cf22c9-11b6-45d5-82d9-c4dc790a768b",
  "performedAt": "2025-08-17T10:00:00.000Z",
  "title": "Push Day (Updated)",
  "notes": "Incline focus",
  "entries": []
}
```

Entry POST Request/Response

```
###
# Create entry #1
Send Request
POST {{baseUrl}}/entries
Content-Type: application/json

{
  "workoutId": "{{workoutId}}",
  "exercise": "Bench Press",
  "sets": 5,
  "reps": 5,
  "weight": 185,
  "restSeconds": 120
}
```

```
HTTP/1.1 201 Created
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 170
ETag: W/"aa-CAnWDpEARGWEsm29vuoplzELIRU"
Date: Mon, 18 Aug 2025 00:52:18 GMT
Connection: close

{
  "id": "d0a5aff0-49c8-48e5-9f7e-cb5470e5f50a",
  "workoutId": "58cf22c9-11b6-45d5-82d9-c4dc790a768b",
  "exercise": "Bench Press",
  "sets": 5,
  "reps": 5,
  "weight": 185,
  "restSeconds": 120
}
```

Challenges and Solutions

DB Connection Issues: Backend spammed retries when starting up, TypeORM couldn't connect

- Fixed it by standardizing DB vars in .env file

```
DATABASE_URL=postgres://postgres:postgres@lo
NODE_ENV=development
```

Choosing Guard: Implementing a useful Guard in backend. Originally included an API key to be sent with write requests

- Decided on simpler Guard required write operations to have content in form of JSON

```
@Injectable()
export class ApiKeyGuard implements CanActivate {
  canActivate(ctx: ExecutionContext) {
    const req = ctx.switchToHttp().getRequest();
    if (['POST','PUT','PATCH','DELETE'].includes(req.method)) {
      return req.header('x-api-key') === process.env.API_KEY;
    }
    return true;
  }
}
```

Original api-key guard

Conclusion and Future Improvements

What I Learned:

- Backend Implementation (NestJS + TypeORM): Controllers/Services, DTO Validation via ValidationPipe and global Guard, entities and relation, Postgres in Docker
- Frontend Implementation (React + TS + Vite): Component-based UI, typed props/events, centralized fetch in APP, dynamic updated after CRUD
- Integrated backend with frontend via CORS
- End-to-end testing with VS Code REST Client

Future Improvements:

- Analytics and charts to track strength progression for each exercise
- Filtering/searches to lookup specific workout/exercise categories
- Include weight tracker to monitor alongside strength progression
- Include daily calorie tracking