==============================

FULL CODEBASE EXPORT (ALL FILES)

==============================

app/**init**.py

```
(The file is empty)
```

---

app/database.py

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

SQLALCHEMY_DATABASE_URL = "sqlite:///./rehab.db"

# Needed for SQLite in single-threaded dev servers
engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args=
{"check_same_thread": False})

SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
Base = declarative_base()

# Dependency for FastAPI routes
def get_db():
        db = SessionLocal()
        try:
                yield db
        finally:
                db.close()
```

---

app/main.py

```python
from fastapi import FastAPI
from .database import Base, engine
from . import models
from .routers import exercises, sessions
from fastapi.staticfiles import StaticFiles
from fastapi.responses import HTMLResponse
from fastapi.responses import RedirectResponse
```

```
# Create tables
Base.metadata.create_all(bind=engine)

app = FastAPI(title="Knee Rehab Habit Tracker", version="0.1.0")

app.include_router(exercises.router)
app.include_router(sessions.router)


@app.get("/")
@app.get("/", include_in_schema=False)
def root():
    return RedirectResponse(url="/ui")

from fastapi.staticfiles import StaticFiles
from fastapi.responses import HTMLResponse

app.mount("/static", StaticFiles(directory="app/static"), name="static")

@app.get("/ui", response_class=HTMLResponse)
def ui_page():
        return """
<!doctype html>
<html lang=\"en\">
<head>
    <meta charset=\"utf-8\">
    <title>Knee Rehab Tracker</title>
    <link rel=\"stylesheet\" href=\"/static/styles.css?v=4\">
</head>
<body>
    <header>
        <div style=\"display:flex;align-items:center;justify-
content:center;gap:1.2em;\">
            <div style=\"background:#fff;border-
radius:50%;padding:0.7em;box-shadow:0 2px 8px
rgba(25,118,210,0.13);display:flex;align-items:center;justify-
content:center;\">
                <svg width=\"36\" height=\"36\" fill=\"none\" viewBox=\"0
0 24 24\"><circle cx=\"12\" cy=\"12\" r=\"12\" fill=\"#1976d2\"/><path
d=\"M12 7v5l4 2\" stroke=\"#fff\" stroke-width=\"2\" stroke-
linecap=\"round\" stroke-linejoin=\"round\"/></svg>
            </div>
            <div>
                <h1 style=\"margin:0;font-size:2.1rem;font-
weight:700;letter-spacing:0.02em;\">Knee Rehab Tracker</h1>
                <div class=\"subtitle\">Your recovery journey</div>
            </div>
        </div>
    </header>
    <nav>
        <button class=\"nav-btn active\" id=\"nav-exercises\"><span>
🏋️</span> Exercises</button>
        <button class=\"nav-btn\" id=\"nav-sessions\"><span>📅</span>
Sessions</button>
```

```
          <button class=\"nav-btn\" id=\"nav-stats\"><span>📊 </span>
Statistics</button>
    </nav>
    <div class=\"main-container\">
    <section class=\"card\" id=\"card-exercises\">
            <div class=\"section-title\">Exercise Library <span
class=\"accent\"></span></div>
            <form id="exerciseForm">
                <div class="grid" style="grid-template-columns: repeat(3,
minmax(0,1fr)); gap: 1.2em 2em; margin-bottom: 0.5em;">
                    <label>Name
                        <input id="name" required placeholder="Heel
Slides">
                    </label>
                    <label>Side
                        <select id="side">
                            <option value="left">left</option>
                            <option value="right">right</option>
                            <option value="both">both</option>
                        </select>
                    </label>
                    <label>Category
                        <select id="category">
                            <option value="strength">strength</option>
                            <option value="mobility"
selected>mobility</option>
                            <option value="balance">balance</option>
                        </select>
                    </label>
                </div>
                <div class="grid" style="grid-template-columns: repeat(3,
minmax(0,1fr)); gap: 1.2em 2em; margin-bottom: 0.5em;">
                    <label>Target sets
                        <input id="target_sets" type="number" min="0"
value="3">
                    </label>
                    <label>Target reps
                        <input id="target_reps" type="number" min="0"
value="12">
                    </label>
                    <label>Target hold (sec)
                        <input id="target_hold_sec" type="number" min="0"
value="2">
                    </label>
                </div>
                <fieldset class="dow">
                    <legend>Days of Week</legend>
                    <label><input type="checkbox" name="dow" value="0">
Sun</label>
                    <label><input type="checkbox" name="dow" value="1">
Mon</label>
                    <label><input type="checkbox" name="dow" value="2">
Tue</label>
                    <label><input type="checkbox" name="dow" value="3">
```

```
Wed</label>
                    <label><input type="checkbox" name="dow" value="4">
Thu</label>
                    <label><input type="checkbox" name="dow" value="5">
Fri</label>
                    <label><input type="checkbox" name="dow" value="6">
Sat</label>
                </fieldset>
                <button type="submit" class="btn-primary"
id="exerciseFormSubmit">Add Exercise</button>
                <button type="button" class="btn-outline"
id="exerciseFormCancel" style="margin-left:10px">Cancel</button>
                <span id="msg" class="msg"></span>
            </form>
        </section>
    <section class=\"card\" id=\"card-exercise-table\">
            <div class=\"section-title\">Exercises <span class=\"accent\">
</span></div>
            <table id=\"exerciseTable\" class=\"table\">
                <thead>
                    <tr>
                        <th>ID</th><th>Name</th><th>Side</th>
<th>Category</th>
                        <th>Targets</th><th>Schedule</th><th>Actions</th>
                    </tr>
                </thead>
                <tbody></tbody>
            </table>
        </section>
    <section class=\"card\" id=\"card-quicklog\">
            <div class=\"section-title\">Quick Log <span class=\"accent\">
</span></div>
            <form id="sessionForm">
                <div class="grid" style="grid-template-columns: repeat(2,
minmax(0,1fr)); gap: 1.2em 2em; margin-bottom: 0.5em;">
                    <label>Date
                        <input id="session_date" type="date" required>
                    </label>
                    <label>Exercise
                        <select id="session_exercise" required></select>
                    </label>
                </div>
                <div class="grid" style="grid-template-columns: repeat(2,
minmax(0,1fr)); gap: 1.2em 2em; margin-bottom: 0.5em;">
                    <label>Sets
                        <input id="session_sets" type="number" min="0">
                    </label>
                    <label>Reps
                        <input id="session_reps" type="number" min="0">
                    </label>
                </div>
                <div class="grid" style="grid-template-columns: repeat(3,
minmax(0,1fr)); gap: 1.2em 2em; margin-bottom: 0.5em;">
                    <label>Hold (sec)
```

```
                        <input id="session_hold_sec" type="number"
min="0">
                    </label>
                    <label>Pain (0-10)
                        <input id="session_pain" type="number" min="0"
max="10">
                    </label>
                    <label>ROM (degrees)
                        <input id="session_rom" type="number" min="0"
max="180">
                    </label>
                </div>
                <button type="submit" class="btn-primary"
id="sessionFormSubmit">Log Session</button>
                <button type="button" class="btn-outline"
id="sessionFormCancel" style="margin-left:10px">Cancel</button>
                <span id="session_msg" class="msg"></span>
            </form>
        </section>
    <section class=\"card\" id=\"card-sessions\">
            <div class=\"section-title\">Session History <span
class=\"accent\"></span></div>
            <table id=\"sessionTable\" class=\"table\">
                <thead>
                    <tr>
                        <th>Date</th><th>Exercise</th><th>Sets</th>
<th>Reps</th><th>Hold</th><th>Pain</th><th>ROM</th><th>Actions</th>
                    </tr>
                </thead>
                <tbody></tbody>
            </table>
        </section>
    <section class=\"card\" id=\"card-stats\">
            <div class=\"section-title\">Stats / Progress <span
class=\"accent\"></span></div>
            <div style=\"display:flex;gap:2em;align-items:flex-
start;justify-content:center;flex-wrap:wrap;\">
                <div style=\"flex:1 1 350px;max-width:420px;\">
                    <canvas id=\"categoryPieChart\" width=\"400\"
height=\"300\"></canvas>
                </div>
                <div style=\"flex:1 1 350px;max-width:520px;\">
                    <h3 style=\"margin-bottom:0.5em;font-
size:1.1em;\">Pain Over Time</h3>
                    <canvas id=\"painLineChart\" width=\"400\"
height=\"300\"></canvas>
                </div>
            </div>
        </section>
    </div>
    <script src=\"https://cdn.jsdelivr.net/npm/chart.js\"></script>
    <script src=\"/static/ui.js?v=4\"></script>
    <script>
        // Tab navigation: scroll to card and highlight active
        document.addEventListener('DOMContentLoaded', function() {
            const navs = [
                {btn: 'nav-exercises', card: 'card-exercises'},
                {btn: 'nav-sessions', card: 'card-sessions'},
                {btn: 'nav-stats', card: 'card-stats'}
            ];
            navs.forEach((nav) => {
                const btn = document.getElementById(nav.btn);
                const card = document.getElementById(nav.card);
                btn.addEventListener('click', function() {
                    // Remove active from all
                    document.querySelectorAll('.nav-btn').forEach(b =>
b.classList.remove('active'));
                    btn.classList.add('active');
                    // Scroll to card
                    if (card) {
                        card.scrollIntoView({behavior: 'smooth', block:
'start'});
                    }
                });
            });
        });
    </script>
</body>
</html>
        """
```

---

app/models.py

```
from sqlalchemy import Column, Integer, String, Text, Date, ForeignKey,
DateTime
from sqlalchemy.orm import relationship
from datetime import datetime
from .database import Base

class Exercise(Base):
    __tablename__ = "exercises"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(120), nullable=False, unique=False, index=True)
    side = Column(String(10), nullable=False)
    category = Column(String(20), nullable=False)
    target_sets = Column(Integer, nullable=True)
    target_reps = Column(Integer, nullable=True)
    target_hold_sec = Column(Integer, nullable=True)
    schedule_dow = Column(Text, nullable=False, default="[]")
    created_at = Column(DateTime, default=datetime.utcnow)

    sessions = relationship("ExerciseSession", back_populates="exercise",
cascade="all, delete")
```

```
class ExerciseSession(Base):
    __tablename__ = "sessions"

    id = Column(Integer, primary_key=True, index=True)
    /* FULL CSS CODE BELOW */
    :root {
    date = Column(Date, nullable=False)
    sets = Column(Integer, nullable=True)
    reps = Column(Integer, nullable=True)
    hold_sec = Column(Integer, nullable=True)
    pain_0_10 = Column(Integer, nullable=True)
    rom_deg = Column(Integer, nullable=True)
    notes = Column(Text, nullable=True)
    created_at = Column(DateTime, default=datetime.utcnow)

    exercise = relationship("Exercise", back_populates="sessions")
```

---

app/schemas.py

```
from typing import Optional, List, Literal
from pydantic import BaseModel, Field, conint
import datetime

PainInt = Optional[conint(ge=0, le=10)]
ROMInt = Optional[conint(ge=0, le=180)]
from typing import Optional, List, Literal
from pydantic import BaseModel, Field, conint
from datetime import date

Side = Literal["left", "right", "both"]
Category = Literal["strength", "mobility", "balance"]

class ExerciseBase(BaseModel):
    name: str = Field(min_length=2, max_length=120)
    side: Side
    category: Category
    target_sets: Optional[int] = None
    target_reps: Optional[int] = None
    target_hold_sec: Optional[int] = None
    # 0=Sun, 1=Mon, ... 6=Sat
    schedule_dow: List[int] = Field(default_factory=list)

class ExerciseCreate(ExerciseBase):
    pass

class ExerciseUpdate(BaseModel):
    name: Optional[str] = None
    side: Optional[Side] = None
    category: Optional[Category] = None
    target_sets: Optional[int] = None
    target_reps: Optional[int] = None
    target_hold_sec: Optional[int] = None
    schedule_dow: Optional[List[int]] = None

class ExerciseOut(ExerciseBase):
    id: int
    class Config:
        from_attributes = True

class SessionBase(BaseModel):
    exercise_id: int
    date: date
    sets: Optional[int] = None
    reps: Optional[int] = None
    hold_sec: Optional[int] = None
    pain_0_10: PainInt = None
    rom_deg: ROMInt = None
class SessionUpdate(BaseModel):
    exercise_id: Optional[int] = None
    date: Optional[datetime.date] = None
    sets: Optional[int] = None
    reps: Optional[int] = None
    hold_sec: Optional[int] = None
    pain_0_10: PainInt = None
    rom_deg: ROMInt = None

class SessionCreate(SessionBase):
    pass

class SessionOut(SessionBase):
    id: int
    class Config:
        from_attributes = True
```

---

app/routers/exercises.py

```
import json
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from ..database import get_db
from .. import models, schemas

router = APIRouter(prefix="/exercises", tags=["exercises"])

@router.post("", response_model=schemas.ExerciseOut)
def create_exercise(payload: schemas.ExerciseCreate, db: Session =
Depends(get_db)):
    ex = models.Exercise(
        name=payload.name,
        side=payload.side,
        category=payload.category,
        target_sets=payload.target_sets,
        target_reps=payload.target_reps,
        target_hold_sec=payload.target_hold_sec,
        schedule_dow=json.dumps(payload.schedule_dow or [])
    )
    db.add(ex)
    db.commit()
    db.refresh(ex)
    # Convert back to list for response
    out = schemas.ExerciseOut(
        id=ex.id,
        name=ex.name,
        side=ex.side,
        category=ex.category,
        target_sets=ex.target_sets,
        target_reps=ex.target_reps,
        target_hold_sec=ex.target_hold_sec,
        schedule_dow=json.loads(ex.schedule_dow or "[]"),
    )
    return out

@router.get("", response_model=list[schemas.ExerciseOut])
def list_exercises(db: Session = Depends(get_db)):
    items = db.query(models.Exercise).order_by(models.Exercise.id).all()
    result = []
    for ex in items:
        result.append(
            schemas.ExerciseOut(
                id=ex.id,
                name=ex.name,
                side=ex.side,
                category=ex.category,
                target_sets=ex.target_sets,
                target_reps=ex.target_reps,
                target_hold_sec=ex.target_hold_sec,
                schedule_dow=json.loads(ex.schedule_dow or "[]"),
            )
        )
    return result

@router.get("/{exercise_id}", response_model=schemas.ExerciseOut)
def get_exercise(exercise_id: int, db: Session = Depends(get_db)):
    ex = db.query(models.Exercise).get(exercise_id)
    if not ex:
        raise HTTPException(status_code=404, detail="Exercise not found")
    return schemas.ExerciseOut(
        id=ex.id,
        name=ex.name,
        side=ex.side,
        category=ex.category,
        target_sets=ex.target_sets,
        target_reps=ex.target_reps,
        target_hold_sec=ex.target_hold_sec,
        schedule_dow=json.loads(ex.schedule_dow or "[]"),
    )

@router.put("/{exercise_id}", response_model=schemas.ExerciseOut)
def update_exercise(exercise_id: int, payload: schemas.ExerciseUpdate, db: 
Session = Depends(get_db)):
    ex = db.query(models.Exercise).get(exercise_id)
    if not ex:
        raise HTTPException(status_code=404, detail="Exercise not found")
    if payload.name is not None: ex.name = payload.name
    if payload.side is not None: ex.side = payload.side
    if payload.category is not None: ex.category = payload.category
    if payload.target_sets is not None: ex.target_sets = 
payload.target_sets
    if payload.target_reps is not None: ex.target_reps = 
payload.target_reps
    if payload.target_hold_sec is not None: ex.target_hold_sec = 
payload.target_hold_sec
    if payload.schedule_dow is not None: ex.schedule_dow = 
json.dumps(payload.schedule_dow)
    db.commit()
    db.refresh(ex)
    return schemas.ExerciseOut(
        id=ex.id,
        name=ex.name,
        side=ex.side,
        category=ex.category,
        target_sets=ex.target_sets,
        target_reps=ex.target_reps,
        target_hold_sec=ex.target_hold_sec,
        schedule_dow=json.loads(ex.schedule_dow or "[]"),
    )

@router.delete("/{exercise_id}", status_code=204)
def delete_exercise(exercise_id: int, db: Session = Depends(get_db)):
    ex = db.query(models.Exercise).get(exercise_id)
    if not ex:
        raise HTTPException(status_code=404, detail="Exercise not found")
    db.delete(ex)
    db.commit()
```

---

app/routers/sessions.py

```
from fastapi import APIRouter, Depends, HTTPException, Query, status
from sqlalchemy.orm import Session
from datetime import date
from ..database import get_db
from .. import models, schemas
```

```
router = APIRouter(prefix="/sessions", tags=["sessions"])

@router.get("/{id}", response_model=schemas.SessionOut)
def get_session(id: int, db: Session = Depends(get_db)):
    s = db.query(models.ExerciseSession).get(id)
    if not s:
        raise HTTPException(status_code=404, detail="Session not found")
    return s

@router.put("/{id}", response_model=schemas.SessionOut)
def update_session(id: int, payload: schemas.SessionUpdate, db: Session =
Depends(get_db)):
    s = db.query(models.ExerciseSession).get(id)
    if not s:
        raise HTTPException(status_code=404, detail="Session not found")
    update_data = payload.model_dump(exclude_unset=True)
    for field, value in update_data.items():
        setattr(s, field, value)
    db.commit()
    db.refresh(s)
    return s

@router.delete("/{id}", status_code=204)
def delete_session(id: int, db: Session = Depends(get_db)):
    s = db.query(models.ExerciseSession).get(id)
    if not s:
        raise HTTPException(status_code=404, detail="Session not found")
    db.delete(s)
    db.commit()

@router.post("", response_model=schemas.SessionOut)
def create_session(payload: schemas.SessionCreate, db: Session =
Depends(get_db)):
    # Ensure exercise exists
    if not db.query(models.Exercise).get(payload.exercise_id):
        raise HTTPException(status_code=400, detail="Exercise does not
exist")
    s = models.ExerciseSession(**payload.model_dump())
    db.add(s)
    db.commit()
    db.refresh(s)
    return s

@router.get("", response_model=list[schemas.SessionOut])
def list_sessions(
    db: Session = Depends(get_db),
    from_date: date = Query(default=None),
    to_date: date = Query(default=None),
    exercise_id: int | None = Query(default=None)
):
    q = db.query(models.ExerciseSession)
    if from_date:
        q = q.filter(models.ExerciseSession.date >= from_date)
    if to_date:
        q = q.filter(models.ExerciseSession.date <= to_date)
    if exercise_id:
        q = q.filter(models.ExerciseSession.exercise_id == exercise_id)
    return q.order_by(models.ExerciseSession.date.desc(),
models.ExerciseSession.id.desc()).all()
```

---

app/static/styles.css

```
:root {
    --primary-blue: #1976d2;
    --primary-blue-light: #2196f3;
    --accent-green: #4caf50;
    --accent-orange: #ff9800;
    --bg-light: #f5f7fa;
    --bg-gray: #e0e3e7;
    --white: #fff;
    --text-main: #222;
    --text-muted: #666;
    --shadow: 0 2px 8px rgba(25, 118, 210, 0.08);
    --radius: 16px;
    --transition: 0.3s cubic-bezier(.4,0,.2,1);
    --font-main: 'Inter', 'Segoe UI', Arial, sans-serif;
}
... (CSS content truncated for brevity) ...
```

---

app/static/ui.js

```
// --- Pain Line Chart ---
let painLineChartInstance = null;
async function renderPainLineChart() {
  const res = await fetch('/sessions');
  const sessions = await res.json();
  const exRes = await fetch('/exercises');
  const exercises = await exRes.json();
  const exMap = {};
  exercises.forEach(ex => { exMap[ex.id] = ex.name; });
  const points = [];
  sessions.forEach(s => {
    if (s.pain_0_10 !== null && s.pain_0_10 !== undefined) {
      points.push({
        x: exMap[s.exercise_id] || `Exercise ${s.exercise_id}`,
        y: s.pain_0_10,
        date: s.date
      });
    }
  });
  // Sort by exercise date
```

```
    points.sort((a, b) => a.x.localeCompare(b.x) ||
  a.date.localeCompare(b.date));
    const ctx = document.getElementById('painLineChart');
    if (!ctx) return;
    if (painLineChartInstance) painLineChartInstance.destroy();
    painLineChartInstance = new Chart(ctx, {
      type: 'line',
      data: {
        labels: points.map(p => p.x),
        datasets: [{
          label: 'Pain (0–10) per Session',
          data: points.map(p => p.y),
          borderColor: chartColor(0),
          backgroundColor: chartColor(0, 0.2),
          tension: 0.2,
          spanGaps: true,
          pointRadius: 4,
          pointHoverRadius: 6
        }]
      },
      options: {
        responsive: false,
        plugins: {
          legend: { display: false },
          title: { display: true, text: 'Pain (0–10) per Exercise' },
          tooltip: {
            callbacks: {
              title: (items) => {
                const idx = items[0].dataIndex;
                return points[idx].x + ' (' + points[idx].date + ')';
              },
              label: (item) => 'Pain: ' + item.formattedValue
            }
          }
        },
        scales: {
          x: { type: 'category', title: {display:true, text:'Exercise'} },
          y: { min: 0, max: 10, title: {display:true, text:'Pain (0–10)'} }
        }
      }
    });
  }
  // --- Pain Line Chart ---
  let painLineChartInstance = null;
  async function renderPainLineChart() {
      const res = await fetch('/sessions');
      const sessions = await res.json();
      const exRes = await fetch('/exercises');
      const exercises = await exRes.json();
      const exMap = {};
      exercises.forEach(ex => { exMap[ex.id] = ex.name; });
      const points = [];
      sessions.forEach(s => {
          if (s.pain_0_10 !== null && s.pain_0_10 !== undefined) {
              points.push({
                  x: exMap[s.exercise_id] || `Exercise ${s.exercise_id}`,
                  y: s.pain_0_10,
                  date: s.date
              });
          }
      });
      // Sort by exercise date
      points.sort((a, b) => a.x.localeCompare(b.x) ||
  a.date.localeCompare(b.date));
      const ctx = document.getElementById('painLineChart');
      if (!ctx) return;
      if (painLineChartInstance) painLineChartInstance.destroy();
      painLineChartInstance = new Chart(ctx, {
          type: 'line',
          data: {
              labels: points.map(p => p.x),
              datasets: [{
                  label: 'Pain (0-10) per Session',
                  data: points.map(p => p.y),
                  borderColor: chartColor(0),
                  backgroundColor: chartColor(0, 0.2),
                  tension: 0.2,
                  spanGaps: true,
                  pointRadius: 4,
                  pointHoverRadius: 6
              }]
          },
          options: {
              responsive: false,
              plugins: {
                  legend: { display: false },
                  title: { display: true, text: 'Pain (0-10) per Exercise'
  },
                  tooltip: {
                      callbacks: {
                          title: (items) => {
                              const idx = items[0].dataIndex;
                              return points[idx].x + ' (' + points[idx].date
  + ')';
                          },
                          label: (item) => 'Pain: ' + item.formattedValue
                      }
                  }
              },
              scales: {
                  x: { type: 'category', title: {display:true,
  text:'Exercise'} },
                  y: { min: 0, max: 10, title: {display:true, text:'Pain (0-
  10)'} }
              }
          }
      });
  }
```

```
// Helper for distinct colors
function chartColor(idx, alpha=1) {
    const palette = [
        '54,162,235', // blue
        '255,99,132', // red
        '255,206,86', // yellow
        '75,192,192', // teal
        '153,102,255', // purple
        '255,159,64', // orange
        '201,203,207' // gray
    ];
    const c = palette[idx % palette.length];
    return `rgba(${c},${alpha})`;
}
// --- Session actions ---
async function deleteSession(id) {
    const ok = confirm('Delete session #' + id + '?');
    if (!ok) return;
    const res = await fetch(`/sessions/${id}`, { method: 'DELETE' });
    if (res.status === 204) {
        await fetchSessions();
    } else {
        alert('Failed to delete (status ' + res.status + ')');
    }
}

let editingSessionId = null;
async function editSession(id) {
    const res = await fetch(`/sessions/${id}`);
    if (!res.ok) {
        alert('Failed to fetch session');
        return;
    }
    const s = await res.json();
    document.getElementById('session_date').value = s.date;
    document.getElementById('session_exercise').value = s.exercise_id;
    document.getElementById('session_sets').value = s.sets ?? '';
    document.getElementById('session_reps').value = s.reps ?? '';
    document.getElementById('session_hold_sec').value = s.hold_sec ?? '';
    document.getElementById('session_pain').value = s.pain_0_10 ?? '';
    document.getElementById('session_rom').value = s.rom_deg ?? '';
    editingSessionId = id;
    document.getElementById('sessionFormSubmit').textContent = 'Update
Session';
    document.getElementById('session_msg').textContent = 'Editing #' + id;
}
// Handle both create and update operations for sessions
async function createSession(ev) {
    ev.preventDefault();
    const romValue = document.getElementById('session_rom').value;

    let payload = {
        exercise_id:
Number(document.getElementById('session_exercise').value),
        date: document.getElementById('session_date').value,
        sets: Number(document.getElementById('session_sets').value) ||
null,
        reps: Number(document.getElementById('session_reps').value) ||
null,
        hold_sec:
Number(document.getElementById('session_hold_sec').value) || null,
        pain_0_10: Number(document.getElementById('session_pain').value)
|| null,
        rom_deg: romValue ? Number(romValue) : null
    };
    console.log('ROM value from form:', romValue);
    console.log('Payload being sent:', payload);
    // For update, only send non-null optional fields (keep exercise_id
and date always)
    if (editingSessionId !== null) {
        // Keep exercise_id and date, only remove null optional fields
        const optionalFields = ['sets', 'reps', 'hold_sec', 'pain_0_10',
'rom_deg'];
        optionalFields.forEach(field => {
            if (payload[field] === null || payload[field] === '' ||
payload[field] === undefined) {
                delete payload[field];
            }
        });
    }
    let url = '/sessions';
    let method = 'POST';
    if (editingSessionId !== null) {
        url = `/sessions/${editingSessionId}`;
        method = 'PUT';
    }
    const res = await fetch(url, {
        method,
        headers: {'Content-Type':'application/json'},
        body: JSON.stringify(payload)
    });
    if (res.ok) {
        document.getElementById('sessionForm').reset();
        document.getElementById('session_msg').textContent =
editingSessionId === null ? 'Session entry added ✓' : 'Session updated ✓';
        editingSessionId = null;
        document.getElementById('sessionFormSubmit').textContent = 'Log
Session';
        fetchSessions();
    } else {
        const txt = await res.text();
        document.getElementById('session_msg').textContent = 'Error: ' +
txt;
    }
    return false;
}
function resetSessionForm() {
    document.getElementById('sessionForm').reset();
    editingSessionId = null;
    document.getElementById('sessionFormSubmit').textContent = 'Log
Session';
    document.getElementById('session_msg').textContent = '';
}
window.addEventListener('DOMContentLoaded', () => {
    let cancelBtn = document.getElementById('sessionFormCancel');
    if (cancelBtn) {
        cancelBtn.onclick = resetSessionForm;
    }
});
async function fetchExercises() {
    const res = await fetch('/exercises');
    const data = await res.json();
    const tbody = document.querySelector('#exerciseTable tbody');
    tbody.innerHTML = '';
    data.forEach(ex => {
        const tr = document.createElement('tr');
        const targets = `${ex.target_sets ?? '-'}×${ex.target_reps ?? '-'}
@ ${ex.target_hold_sec ?? 0}s`;
        const schedule = (ex.schedule_dow || []).sort().join(', ');
        tr.innerHTML = `
            <td>${ex.id}</td>
            <td>${ex.name}</td>
            <td><span class="badge">${ex.side}</span></td>
            <td><span class="badge">${ex.category}</span></td>
            <td>${targets}</td>
            <td>${schedule}</td>
            <td>
                <button class="action-del"
onclick="deleteExercise(${ex.id})">Delete</button>
                <button onclick="editExercise(${ex.id})">Edit</button>
            </td>
        `;
        tbody.appendChild(tr);
    });
    // Populate session exercise dropdown
    const sessionExercise = document.getElementById('session_exercise');
    if (sessionExercise) {
        sessionExercise.innerHTML = '';
        data.forEach(ex => {
            const opt = document.createElement('option');
            opt.value = ex.id;
            opt.textContent = ex.name;
            sessionExercise.appendChild(opt);
        });
    }
    // --- Pie chart for category distribution ---
    renderCategoryPieChart(data);
}
// Pie chart rendering for exercise category distribution
let categoryPieChartInstance = null;
function renderCategoryPieChart(exercises) {
    const ctx = document.getElementById('categoryPieChart');
    if (!ctx) return;
    // Count categories
    const counts = { strength: 0, mobility: 0, balance: 0 };
    exercises.forEach(ex => {
        if (counts[ex.category] !== undefined) counts[ex.category]++;
    });
    const labels = ['Strength', 'Mobility', 'Balance'];
    const data = [counts.strength, counts.mobility, counts.balance];
    // Destroy previous chart if exists
    if (categoryPieChartInstance) {
        categoryPieChartInstance.destroy();
    }
    categoryPieChartInstance = new Chart(ctx, {
        type: 'pie',
        data: {
            labels: labels,
            datasets: [{
                data: data,
                backgroundColor: [
                    'rgba(54, 162, 235, 0.7)', // strength
                    'rgba(255, 206, 86, 0.7)', // mobility
                    'rgba(75, 192, 192, 0.7)'  // balance
                ],
                borderColor: [
                    'rgba(54, 162, 235, 1)',
                    'rgba(255, 206, 86, 1)',
                    'rgba(75, 192, 192, 1)'
                ],
                borderWidth: 1
            }]
        },
        options: {
            responsive: false,
            plugins: {
                legend: {
                    display: true,
                    position: 'bottom'
                },
                title: {
                    display: true,
                    text: 'Exercise Category Distribution'
                }
            }
        }
    });
}
async function deleteExercise(id) {
    const ok = confirm('Delete exercise #' + id + '?');
    if (!ok) return;
    const res = await fetch(`/exercises/${id}`, { method: 'DELETE' });
    if (res.status === 204) {
        await fetchExercises();
    } else {
        alert('Failed to delete (status ' + res.status + ')');
    }
}
function readCheckedDOW() {
    return
Array.from(document.querySelectorAll('input[name="dow"]:checked'))
        .map(cb => Number(cb.value))
        .sort((a,b)=>a-b);
}
let editingExerciseId = null;
async function createOrUpdateExercise(ev) {
    ev.preventDefault();
    const payload = {
        name: document.getElementById('name').value.trim(),
        side: document.getElementById('side').value,
        category: document.getElementById('category').value,
        target_sets: Number(document.getElementById('target_sets').value)
|| null,
        target_reps: Number(document.getElementById('target_reps').value)
|| null,
        target_hold_sec:
Number(document.getElementById('target_hold_sec').value) || null,
        schedule_dow: readCheckedDOW()
    };
    let url = '/exercises';
    let method = 'POST';
    if (editingExerciseId !== null) {
        url = `/exercises/${editingExerciseId}`;
        method = 'PUT';
    }
    const res = await fetch(url, {
        method,
        headers: {'Content-Type':'application/json'},
        body: JSON.stringify(payload)
    });
    if (res.ok) {
        document.getElementById('exerciseForm').reset();
        document.getElementById('msg').textContent = editingExerciseId ===
null ? 'Saved ✓' : 'Updated ✓';
        editingExerciseId = null;
        document.getElementById('exerciseFormSubmit').textContent = 'Add
Exercise';
        fetchExercises();
    } else {
        const txt = await res.text();
        document.getElementById('msg').textContent = 'Error: ' + txt;
    }
    return false;
}
async function editExercise(id) {
    const res = await fetch(`/exercises/${id}`);
    if (!res.ok) {
        alert('Failed to fetch exercise');
        return;
    }
    const ex = await res.json();
    document.getElementById('name').value = ex.name;
    document.getElementById('side').value = ex.side;
    document.getElementById('category').value = ex.category;
    document.getElementById('target_sets').value = ex.target_sets ?? '';
    document.getElementById('target_reps').value = ex.target_reps ?? '';
    document.getElementById('target_hold_sec').value = ex.target_hold_sec
?? '';
    // Uncheck all DOW checkboxes first
    document.querySelectorAll('input[name="dow"]').forEach(cb => {
cb.checked = false; });
    (ex.schedule_dow || []).forEach(dow => {
        const cb = document.querySelector(`input[name="dow"]
[value="${dow}"]`);
        if (cb) cb.checked = true;
    });
    editingExerciseId = id;
    document.getElementById('exerciseFormSubmit').textContent = 'Update
Exercise';
    document.getElementById('msg').textContent = 'Editing #' + id;
}
function resetExerciseForm() {
    document.getElementById('exerciseForm').reset();
    editingExerciseId = null;
    document.getElementById('exerciseFormSubmit').textContent = 'Add
Exercise';
    document.getElementById('msg').textContent = '';
}
window.addEventListener('DOMContentLoaded', () => {
    fetchExercises();
    // Attach new handler for form submit
    const form = document.getElementById('exerciseForm');
    if (form) {
        form.onsubmit = createOrUpdateExercise;
    }
    // Always attach cancel handler to Cancel button if present
    let cancelBtn = document.getElementById('exerciseFormCancel');
    if (cancelBtn) {
        cancelBtn.onclick = resetExerciseForm;
    } else if (form) {
        // fallback: add if not present
        cancelBtn = document.createElement('button');
        cancelBtn.type = 'button';
        cancelBtn.id = 'exerciseFormCancel';
        cancelBtn.textContent = 'Cancel';
        cancelBtn.style.marginLeft = '10px';
        cancelBtn.onclick = resetExerciseForm;
        form.appendChild(cancelBtn);
    }
    // Session form handler
    const sessionForm = document.getElementById('sessionForm');
    if (sessionForm) {
        sessionForm.onsubmit = createSession;
    }
    fetchSessions();
    renderPainLineChart();
});
async function fetchSessions() {
    const res = await fetch('/sessions');
    const data = await res.json();
    // Get exercise names for mapping
    const exRes = await fetch('/exercises');
    const exData = await exRes.json();
    const exMap = {};
    exData.forEach(ex => { exMap[ex.id] = ex.name; });
    const tbody = document.querySelector('#sessionTable tbody');
    tbody.innerHTML = '';
    data.forEach(s => {
        console.log('Session data:', s); // Debug log
        const tr = document.createElement('tr');
        tr.innerHTML = `
            <td>${s.date}</td>
            <td>${exMap[s.exercise_id] || s.exercise_id}</td>
            <td>${s.sets ?? ''}</td>
            <td>${s.reps ?? ''}</td>
            <td>${s.hold_sec ?? ''}</td>
            <td>${s.pain_0_10 ?? ''}</td>
            <td>${s.rom_deg ?? ''}</td>
            <td>
                <button onclick="editSession(${s.id})">Edit</button>
                <button onclick="deleteSession(${s.id})">Delete</button>
            </td>
        `;
        tbody.appendChild(tr);
    });
    // Update chart
    renderPainLineChart();
}
```

```
```