Final Milestone Assessment (60 Marks | 120 Minutes)

Format: User Story + Practical Tasks
Complexity: Average
Tech Stack: MERN (React, Node.js, Express, MongoDB), Validation, Middleware, Basic Testing

---

Problem Statement: FitTrack – Personal Fitness Training Portal

FitTrack is building a basic MERN-based portal where users can explore fitness programs, enroll in one, and manage their enrolled programs. You are hired as a MERN developer to build a minimum viable system that satisfies two major workflow segments.

Segment A – Backend (30 Marks)

User Story A1 – Program Catalog API (15 Marks)

As a platform admin, I want to add and list fitness programs so that users can choose a plan.

Tasks

1. **Create a Mongoose model named Program** with fields:
    - programId (string, unique, required)
    - name (string, required)

- category (string, required)
- level (string: Beginner, Intermediate, Advanced)
- price (number, required, min 0)
- createdAt (date, default now)

2. **API Endpoints (Express Router):**
   - POST /api/programs → add a program
   - GET /api/programs → fetch all programs
3. **Validation using Joi or express-validator** for POST program.
4. **Global Error Handling Middleware**
   - Must catch validation errors
   - Must catch server errors

Must return unified response format:

```
 {
  success: false,
  message: "",
  data: null
}
```

   -
5. Follow REST conventions and proper HTTP codes.

User Story A2 – Enrollment API (15 Marks)

As a user, I want to enroll in a fitness program to start my training journey.

Tasks

1. Create an Enrollment model with fields:
    - userId
    - programId
    - enrolledOn (default date)
2. Implement POST /api/enroll API:
    - Validate userId as required
    - Validate programId exists in Program collection
    - Prevent duplicate enrollment
    - Return 201 on success
3. Write one test using Mocha + Chai or SuperTest:
    - Successful enrollment → 201
    - Duplicate enrollment → 400

Segment B – Frontend (30 Marks)

User Story B1 – Program Listing UI (15 Marks)

As a user, I want to view all available fitness programs so that I can make an informed choice.

Tasks

1. Create a React component ProgramList.js.
2. Fetch data from /api/programs using useEffect.
3. Display program name, category, level, and price in a styled card/list form.
4. Add an “Enroll” button for each program.
5. Implement loading and error UI states.

User Story B2 – Enrollment Interaction UI (15 Marks)

As a user, I want to enroll with a button click and view feedback immediately.

Tasks

1. Clicking “Enroll” sends a POST request to /api/enroll.
2. Display messages:
    - Success
    - Already enrolled
    - API/server error

3. Maintain an enrolledPrograms state.

4. Show enrolled programs below the list in a simple layout.

Rubrics (60 Marks Total)

Backend – 30 Marks

Evaluation Criteria	Marks
Mongoose Model Quality (Program, Enrollment)	5
API Routes & Validation Correctness	10
Error Handling Middleware & Response Structuring	5
Enrollment Logic (duplicate check, linking)	5
API Testing (Mocha/Chai/SuperTest)	5

Frontend – 30 Marks

Evaluation Criteria	Marks
React Component Structure	5
API Integration & Data Fetching	5
UI Rendering & Program Listing	5
Enrollment Flow Functionality	10
Error, Loading States, and UX	5

Submission Format

Submit a ZIP file:

<YourName>_MockAssessment_M3.zip

Include:

Backend:

- Models
- Routes
- Middleware
- Tests
- Postman collection (optional)

Frontend:

- ProgramList.js
- Enrollment components
- API utility file (optional)

Documentation:

- README.md with:
    - Setup instructions
    - How to run backend and frontend
    - List of APIs

- Screenshots

Sample Dataset for Milestone 3 Mock Assessment

The dataset includes:

1. **Program Catalog Sample Data (Programs Collection)**
2. **Optional User Sample Data (Users Collection)** - useful for enrollment testing
3. **Enrollment Sample Data (for reference only)** - do not pre-insert unless needed for a demo

---

1. Programs Collection (programs.json)

```
[
  {
    "programId": "FTP001",
    "name": "Beginner Full Body Workout",
    "category": "Strength Training",
    "level": "Beginner",
    "price": 1999
  },
  {
    "programId": "FTP002",
```

```
  "name": "Weight Loss Intensive",
  "category": "Cardio",
  "level": "Intermediate",
  "price": 2499
},
{
  "programId": "FTP003",
  "name": "HIIT High Performance",
  "category": "Cardio",
  "level": "Advanced",
  "price": 2999
},
{
  "programId": "FTP004",
  "name": "Yoga Mindfulness Program",
  "category": "Yoga",
  "level": "Beginner",
  "price": 1499
},
{
  "programId": "FTP005",
  "name": "Lean Muscle Builder",
```

```
    "category": "Strength Training",
    "level": "Intermediate",
    "price": 2799
  }
]
```

This dataset contains diverse categories and levels to test UI rendering, filtering, validation, and enrollment logic.

---

2. Users Collection (optional but recommended)

You may create a mock user table for testing role-based validations or enrollment.

```
[
  {
    "userId": "USR101",
    "name": "Arjun Mehta",
    "email": "arjun.mehta@example.com"
  },
  {
    "userId": "USR102",
    "name": "Sneha Verma",
    "email": "sneha.verma@example.com"
  }
```

```
]
```

You can use these two users to test:

- Valid enrollment
- Duplicate enrollment
- Error handling

---

3. Enrollment Collection (optional test-only)

```
Do not insert this initially.
 Add it only when testing duplicate logic.

[
  {
    "userId": "USR101",
    "programId": "FTP001",
    "enrolledOn": "2025-01-01T10:30:00Z"
  }
]
```

This helps validate:

- Duplicate detection
- HTTP 400 response

- Preventing duplicate enrollment for same user-program pair

Sample Postman Payloads

Create Program

```
{
  "programId": "FTP006",
  "name": "Athletic Conditioning",
  "category": "Sports Training",
  "level": "Advanced",
  "price": 3499
}
```

Enrollment Request

```
{
  "userId": "USR101",
  "programId": "FTP003"
   }
```