

Real-time Student Task App

Project Title

Student Task Tracker App (Real-Time with MongoDB Atlas)

Core Use Case

An admin (teacher) uploads students (via Excel or manually), assigns tasks to individual students, and tracks their task progress. Each student sees only their tasks, completes them, and their report (with performance graph) is updated in real-time.

Roles

1. Admin (Teacher)

- Upload student data via Excel or manually
- Assign tasks to individual students
- View and export task completion reports
- Delete students
- View top performers

- Communicate with students (optional: messaging)

2. Student

- Login with credentials provided by admin
 - View assigned tasks
 - Mark tasks as completed
 - See personal progress graph
-



App Features



Authentication

- Admin and student login
- Student credentials generated by admin
- Firebase Auth (optional) or JWT-based manual login



Admin Panel Features (App/Web Flutter UI)

- Upload Excel file to import students
- Add/edit/delete student manually
- Assign task to specific student(s)
- View student-wise task reports with completion stats
- View chart-based reports (Bar/Line graph)
- Export student data and task reports
- Delete student + tasks
- View top-performing students



Student Panel Features

- Login with given ID/password
 - See own profile and assigned tasks
 - Mark task status: **Pending** → **Completed**
 - View progress report (Graph)
 - Task completion streaks (optional gamification)
-



Database Structure (MongoDB Atlas)

Collections:

plaintext

1. users (for both admin and students)
2. tasks
3. reports (optional)

users collection

```
json

{
  "_id": "ObjectId",
  "name": "Ali Raza",
  "email": "ali@student.com",
  "password": "hashed_password",
  "role": "student", // or "admin"
  "created_at": "2025-05-12T10:00:00Z"
}
```

tasks collection

```
json

{
  "_id": "ObjectId",
  "title": "Complete UI Design",
  "description": "Design task screen in Flutter",
  "assigned_to": "student_user_id",
  "status": "pending", // or "completed"
  "due_date": "2025-05-20",
  "created_by": "admin_user_id"
}
```

Optional: reports collection

```
json

{
  "student_id": "ObjectId",
  "completed_tasks": 5,
  "pending_tasks": 3,
  "performance_score": 80 // calculated
}
```



Technology Stack

Component	Technology
Frontend	Flutter (for both roles)
Backend API	Node.js + Express
Database	MongoDB Atlas
Authentication	JWT (or Firebase Auth)
Excel Upload	`multer` + `xlsx` (Node.js)
Charts	`fl_chart` (Flutter)

Component	Technology
Hosting (optional)	Render, Railway, or Vercel



Bonus Features (Optional)

- 🔔 Push Notifications when a new task is assigned
- 💬 In-app messaging (for admin-student chat)
- 🏆 Leaderboard screen for students
- 📊 Weekly/monthly report generation
- 📧 Email credentials to students when added



Sample Dashboard Screens

1. Admin Dashboard

- Total Students
- Tasks Assigned Today
- Top 5 Students (Bar Graph)

2. Student Dashboard

- My Tasks
- Task Completion Progress
- Task Calendar



Conclusion


Yes, this app is **very much feasible using Flutter + MongoDB Atlas**, provided you build a **middleware backend** to manage authentication, Excel file parsing, and access control. This architecture is scalable and good for teaching **real-world app development** to your students.

Would you like me to generate a **complete folder structure and API endpoints list** to start development right away?

No problem — let's clarify this:

? Can MongoDB Atlas work directly with Flutter?

Short answer:

 **No**, you cannot connect **Flutter directly to MongoDB Atlas** on the client side because:

- MongoDB Atlas is a **database**, not an API.
- It doesn't support direct public access from mobile apps due to **security risks** (like exposing connection strings).
- MongoDB requires server-side drivers (like Node.js, Python, etc.).

Correct Way: Use a Backend between Flutter and MongoDB Atlas

You should follow this **architecture**:

```
SCSS

Flutter (Admin & Student Apps)
  ↑↓
  REST API (Node.js + Express)
  ↑↓
  MongoDB Atlas (Cloud Database)
```

Why This Is Required:

- MongoDB Atlas connection string contains **database credentials** — exposing this in a Flutter app would be a **huge security issue**.
- A backend allows you to:
 - Authenticate users (admin/student)
 - Control data access (role-based logic)
 - Handle Excel file uploads
 - Manage database logic safely

What You Need to Build:

Layer	Tech	Responsibility
Frontend	Flutter (2 apps or roles)	UI, user interaction
Backend	Node.js + Express	API endpoints, auth, logic, Excel import
Database	MongoDB Atlas	Store students, tasks, reports

Example API Flow

Action	Flutter Request	API (Node.js) Task
Admin login	<code>`POST /auth/login`</code>	Validate credentials, return JWT
Add student (Excel/manual)	<code>`POST /students/upload`</code>	Parse file, save students in MongoDB
Student login	<code>`POST /auth/login`</code>	Authenticate student
Get student tasks	<code>`GET /tasks/my`</code>	Return only their assigned tasks
Mark task as complete	<code>`POST /tasks/:id/done`</code>	Update task status in DB

Action	Flutter Request	API (Node.js) Task
Generate report	`GET /reports/:id`	Calculate completed vs pending tasks

Would you like me to give you:

1. ☒ A **backend API template (Node.js + MongoDB Atlas)**
2. ☒ A **Flutter folder structure** for Admin and Student roles
3. ☒ Sample API calling code in Flutter (`http` package)

Just let me know how you'd like to proceed!