MP Task API (Node/Express/Mongoose) – Starter Template

This starter implements the assignment requirements you shared:

- Node + Express + Mongoose
- Endpoints: `/api/users` and `/api/tasks` with `GET/POST`, plus `/:id` with `GET/PUT/DELETE`
- JSON-encoded query params: `where`, `sort`, `select`, `skip`, `limit`, `count`
- Response envelope `{ message, data }` with appropriate HTTP status codes
- Server-side validation for required fields and email uniqueness
- Two-way references between `User.pendingTasks` and `Task.assignedUser/assignedUserName`
- MongoDB Atlas connection via `.env`

  How to use

  1. Create a new empty repo and copy these files.
  2. `npm install`
  3. Add your Atlas URI to `.env` (see `.env.example`).
  4. `npm start` and test via the sample requests at the bottom.

Project Structure

```
mp-task-api/
├── package.json
├── .env.example
├── README.md
└── src/
    ├── server.js
    ├── db.js
    ├── models/
    │   ├── User.js
    │   └── Task.js
    ├── routes/
    │   ├── users.js
    │   └── tasks.js
    └── utils/
        ├── query.js
        └── response.js
```

---

package.json

```
{
  "name": "mp-task-api",
  "version": "1.0.0",
  "description": "Task management API (MP spec)",
  "main": "src/server.js",
  "type": "module",
  "scripts": {
    "start": "node src/server.js",
    "dev": "NODE_ENV=development nodemon src/server.js"
  },
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "mongoose": "^8.5.1",
    "morgan": "^1.10.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.0"
  }
}
```

.env.example

```
# Copy to .env and set your Atlas URI
MONGODB_URI=mongodb+srv://<user>:<password>@<cluster>/<db>?retryWrites=true&w=majority
PORT=3000
```

---

src/utils/response.js

```
export function ok(res, data, message = "OK") {
  return res.status(200).json({ message, data });
}

export function created(res, data, message = "Created") {
  return res.status(201).json({ message, data });
}

export function noContent(res, message = "No Content") {
  return res.status(204).json({ message, data: null });
}

export function badRequest(res, message = "Bad Request", data = null) {
  return res.status(400).json({ message, data });
}

export function notFound(res, message = "Not Found", data = null) {
  return res.status(404).json({ message, data });
}

export function serverError(res, err, fallback = "Server Error") {
  const message = err?.message || fallback;
  return res.status(500).json({ message, data: null });
}
```

---

src/utils/query.js

```
// Apply JSON-encoded query string options to a Mongoose query.
// Supports: where, sort, select, skip, limit, count

export function parseJSON(value, fallback) {
  if (value === undefined) return fallback;
  try { return JSON.parse(value); } catch {
    return fallback; // caller should return 400 if needed
  }
}

export function applyQueryParams(model, reqQuery, baseFilter = {}) {
  const q = model.find(baseFilter);

  // where
  if (reqQuery.where !== undefined) {
    const where = parseJSON(reqQuery.where);
    if (where === undefined) throw new Error("Invalid JSON for 'where'.");
    q.find(where);
  }

  // sort
  if (reqQuery.sort !== undefined) {
    const sort = parseJSON(reqQuery.sort);
    if (sort === undefined) throw new Error("Invalid JSON for 'sort'.");
    q.sort(sort);
  }

  // select
  if (reqQuery.select !== undefined) {
    const select = parseJSON(reqQuery.select);
    if (select === undefined) throw new Error("Invalid JSON for 'select'.");
    q.select(select);
  }

  // skip
  if (reqQuery.skip !== undefined) {
    const skip = Number(reqQuery.skip);
    if (Number.isNaN(skip) || skip < 0) throw new Error("Invalid 'skip'.");
    q.skip(skip);
  }

  // limit (note: default handled by routes per MP spec)
  if (reqQuery.limit !== undefined) {
    const limit = Number(reqQuery.limit);
    if (Number.isNaN(limit) || limit < 0) throw new Error("Invalid 'limit'.");
    q.limit(limit);
  }

  // count
  const count = String(reqQuery.count).toLowerCase() === "true";

  return { query: q, count };
}
```

---

src/db.js

```
import mongoose from 'mongoose';

export async function connect(uri) {
  mongoose.set('strictQuery', false);
  await mongoose.connect(uri, {
    serverSelectionTimeoutMS: 10000
  });
}
```

---

src/models/User.js

```
import mongoose from 'mongoose';

const UserSchema = new mongoose.Schema({
  name: { type: String, required: [true, 'User name is required'] },
  email: { type: String, required: [true, 'Email is required'], unique: true, index: true },
  // Store Task ObjectIds. MP says "_id fields of the pending tasks"; refs keep it consistent.
  pendingTasks: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Task' }],
  dateCreated: { type: Date, default: Date.now }
});

export default mongoose.model('User', UserSchema);
```

---

src/models/Task.js

```
import mongoose from 'mongoose';

const TaskSchema = new mongoose.Schema({
  name: { type: String, required: [true, 'Task name is required'] },
  description: { type: String, default: '' },
  deadline: { type: Date, required: [true, 'Deadline is required'] },
  completed: { type: Boolean, default: false },
  assignedUser: { type: mongoose.Schema.Types.ObjectId, ref: 'User', default: null },
  assignedUserName: { type: String, default: 'unassigned' },
  dateCreated: { type: Date, default: Date.now }
});

export default mongoose.model('Task', TaskSchema);
```

---

src/routes/users.js

```
import express from 'express';
import User from '../models/User.js';
import Task from '../models/Task.js';
import { applyQueryParams } from '../utils/query.js';
import { ok, created, badRequest, notFound, serverError } from '../utils/response.js';

const router = express.Router();

// GET /api/users
router.get('/', async (req, res) => {
  try {
    const { query, count } = applyQueryParams(User, req.query);

    // MP: users default has NO limit unless provided
    if (req.query.limit === undefined) {
      // no implicit limit
    }

    if (count) {
      const n = await User.countDocuments(query.getQuery());
      return ok(res, n);
    }

    const docs = await query.exec();
    return ok(res, docs);
  } catch (err) {
    if (err.message.startsWith('Invalid')) return badRequest(res, err.message);
    return serverError(res, err);
  }
```

```
});

// POST /api/users
router.post('/', async (req, res) => {
 try {
  const { name, email, pendingTasks } = req.body || {};
  if (!name && !email) return badRequest(res, 'User must have name or email');
  if (!name) return badRequest(res, 'User name is required');
  if (!email) return badRequest(res, 'User email is required');

  const exists = await User.findOne({ email });
  if (exists) return badRequest(res, 'Email already exists');

  const user = await User.create({ name, email, pendingTasks: pendingTasks || [] });

  // If pendingTasks are provided, ensure two-way linkage
  if (Array.isArray(pendingTasks) && pendingTasks.length) {
   await Task.updateMany(
    { _id: { $in: pendingTasks } },
    { $set: { assignedUser: user._id, assignedUserName: user.name } }
   );
  }

  return created(res, user);
 } catch (err) {
  if (err?.code === 11000) return badRequest(res, 'Email already exists');
  return serverError(res, err);
 }
});

// GET /api/users/:id
router.get('/:id', async (req, res) => {
 try {
  const user = await User.findById(req.params.id);
  if (!user) return notFound(res, 'User not found');
  return ok(res, user);
 } catch (err) { return serverError(res, err); }
});

// PUT /api/users/:id (replace entire user)
router.put('/:id', async (req, res) => {
 try {
  const { name, email, pendingTasks } = req.body || {};

  if (!name && !email) return badRequest(res, 'User must have name or email');
  if (!name) return badRequest(res, 'User name is required');
  if (!email) return badRequest(res, 'User email is required');

  const user = await User.findById(req.params.id);
  if (!user) return notFound(res, 'User not found');
```

```
    // If email changed, ensure uniqueness
    if (email !== user.email) {
      const exists = await User.findOne({ email });
      if (exists) return badRequest(res, 'Email already exists');
    }

    // Unassign all previous pendingTasks from this user
    await Task.updateMany(
      { _id: { $in: user.pendingTasks } },
      { $set: { assignedUser: null, assignedUserName: 'unassigned' } }
    );

    // Apply new fields
    user.name = name;
    user.email = email;
    user.pendingTasks = Array.isArray(pendingTasks) ? pendingTasks : [];
    await user.save();

    // Two-way reference: assign tasks listed in pendingTasks
    if (user.pendingTasks.length) {
      await Task.updateMany(
        { _id: { $in: user.pendingTasks } },
        { $set: { assignedUser: user._id, assignedUserName: user.name } }
      );
    }

    return ok(res, user, 'Updated');
  } catch (err) { return serverError(res, err); }
});

// DELETE /api/users/:id
router.delete('/:id', async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) return notFound(res, 'User not found');

    // Unassign user's pending tasks (two-way requirement)
    await Task.updateMany(
      { _id: { $in: user.pendingTasks } },
      { $set: { assignedUser: null, assignedUserName: 'unassigned' } }
    );

    await user.deleteOne();
    return ok(res, null, 'Deleted');
  } catch (err) { return serverError(res, err); }
});

export default router;
```

src/routes/tasks.js

```
import express from 'express';
import Task from '../models/Task.js';
import User from '../models/User.js';
import { applyQueryParams } from '../utils/query.js';
import { ok, created, badRequest, notFound, serverError } from '../utils/response.js';

const router = express.Router();

// GET /api/tasks
router.get('/', async (req, res) => {
 try {
  const { query, count } = applyQueryParams(Task, req.query);

  // MP: default limit for tasks is 100 unless specified
  if (req.query.limit === undefined) {
   query.limit(100);
  }

  if (count) {
   const n = await Task.countDocuments(query.getQuery());
   return ok(res, n);
  }

  const docs = await query.exec();
  return ok(res, docs);
 } catch (err) {
  if (err.message.startsWith('Invalid')) return badRequest(res, err.message);
  return serverError(res, err);
 }
});

// POST /api/tasks
router.post('/', async (req, res) => {
 try {
  const { name, description, deadline, completed, assignedUser, assignedUserName } = req.body ||
{};
  if (!name) return badRequest(res, 'Task name is required');
  if (!deadline) return badRequest(res, 'Deadline is required');

  const task = await Task.create({
   name,
   description: description || '',
   deadline,
   completed: Boolean(completed),
   assignedUser: assignedUser || null,
   assignedUserName: assignedUserName || (assignedUser ? '' : 'unassigned')
  });
```

```
    // two-way link: if assignedUser provided, push into user's pendingTasks
    if (task.assignedUser) {
      const user = await User.findById(task.assignedUser);
      if (user) {
        user.pendingTasks.addToSet(task._id);
        if (!task.assignedUserName) task.assignedUserName = user.name;
        await user.save();
        await task.save();
      }
    }

    return created(res, task);
  } catch (err) { return serverError(res, err); }
});

// GET /api/tasks/:id
router.get('/:id', async (req, res) => {
  try {
    const task = await Task.findById(req.params.id);
    if (!task) return notFound(res, 'Task not found');
    return ok(res, task);
  } catch (err) { return serverError(res, err); }
});

// PUT /api/tasks/:id (replace entire task)
router.put('/:id', async (req, res) => {
  try {
    const { name, description, deadline, completed, assignedUser, assignedUserName } = req.body ||
{};
    if (!name) return badRequest(res, 'Task name is required');
    if (!deadline) return badRequest(res, 'Deadline is required');

    const task = await Task.findById(req.params.id);
    if (!task) return notFound(res, 'Task not found');

    // Remove from old user's pendingTasks if reassigned or unassigned
    if (task.assignedUser) {
      await User.updateOne({ _id: task.assignedUser }, { $pull: { pendingTasks: task._id } });
    }

    // Apply new fields
    task.name = name;
    task.description = description || '';
    task.deadline = deadline;
    task.completed = Boolean(completed);
    task.assignedUser = assignedUser || null;
    task.assignedUserName = assignedUser ? (assignedUserName || task.assignedUserName || '') :
'unassigned';

    await task.save();
    // Add to new user's pendingTasks
    if (task.assignedUser) {
      const newUser = await User.findById(task.assignedUser);
      if (newUser) {
        newUser.pendingTasks.addToSet(task._id);
        if (!assignedUserName) {
          task.assignedUserName = newUser.name;
          await task.save();
        }
        await newUser.save();
      }
    }

    return ok(res, task, 'Updated');
  } catch (err) { return serverError(res, err); }
});

// DELETE /api/tasks/:id
router.delete('/:id', async (req, res) => {
  try {
    const task = await Task.findById(req.params.id);
    if (!task) return notFound(res, 'Task not found');

    // two-way: remove from assigned user's pendingTasks
    if (task.assignedUser) {
      await User.updateOne({ _id: task.assignedUser }, { $pull: { pendingTasks: task._id } });
    }

    await task.deleteOne();
    return ok(res, null, 'Deleted');
  } catch (err) { return serverError(res, err); }
});

export default router;
```

src/server.js

```
import express from 'express';
import dotenv from 'dotenv';
import morgan from 'morgan';
import cors from 'cors';
import { connect } from './db.js';
import usersRoute from './routes/users.js';
import tasksRoute from './routes/tasks.js';
import { badRequest, ok } from './utils/response.js';

dotenv.config();
```

```
const app = express();
app.use(cors());
app.use(express.json());
app.use(morgan('dev'));

// Health check
app.get('/api', (req, res) => ok(res, { service: 'mp-task-api' }, 'API is up'));

app.use('/api/users', usersRoute);
app.use('/api/tasks', tasksRoute);

// 400 handler for malformed JSON in query strings
app.use((err, req, res, next) => {
  if (err instanceof SyntaxError && 'body' in err) {
    return badRequest(res, 'Invalid JSON body');
  }
  return next(err);
});

const PORT = process.env.PORT || 3000;
connect(process.env.MONGODB_URI)
  .then(() => {
    app.listen(PORT, () => {
      console.log(`Server listening on http://localhost:${PORT}`);
    });
  })
  .catch((e) => {
    console.error('Failed to connect to MongoDB', e);
    process.exit(1);
  });
```

README.md (quick start)

```
# MP Task API

Implements the MP spec with Node, Express, and Mongoose.

## Setup
1. `npm install`
2. Copy `.env.example` to `.env` and set `MONGODB_URI`.
3. `npm start` (defaults to port 3000).

## Endpoints
- `GET /api/users` – list users
- `POST /api/users` – create user `{ name, email, pendingTasks? }`
- `GET /api/users/:id` – fetch user
- `PUT /api/users/:id` – replace user (must include name & email & optional pendingTasks)
- `DELETE /api/users/:id` – delete user (unassigns his pending tasks)
- `GET /api/tasks` – list tasks (default `limit=100`)
- `POST /api/tasks` – create task `{ name, description?, deadline, completed?, assignedUser?, assignedUserName? }`
- `GET /api/tasks/:id` – fetch task
- `PUT /api/tasks/:id` – replace task (must include name & deadline)
- `DELETE /api/tasks/:id` – delete task (removes from assigned user's pendingTasks)

## Query String Parameters
All JSON-encoded: `where`, `sort`, `select`, `skip`, `limit`, `count`.

Examples:
```

GET /api/tasks GET /api/users GET /api/users?where={"_id":"..."} GET /api/tasks?where={"completed"\:true} GET /api/tasks?where={"_id":{"\$in":["id1","id2"]}} GET /api/users?sort={"name":1} GET /api/users?select={"_id":0} GET /api/tasks?skip=60&limit=20

```
## Notes
- Responses always `{ message, data }` with status codes 200/201/204/400/404/500.
- Validations:
  - User requires `name` and `email` (unique).
  - Task requires `name` and `deadline`.
- Two-way references are enforced for all PUT/POST/DELETE as specified.
```

Sample cURL

```
# Health
curl http://localhost:3000/api

# Create user
curl -X POST http://localhost:3000/api/users
 -H 'Content-Type: application/json'
 -d '{"name":"Alice","email":"alice@example.com"}'

# Create task assigned to Alice (replace <USER_ID>)
curl -X POST http://localhost:3000/api/tasks
 -H 'Content-Type: application/json'
 -d '{"name":"Write report","deadline":"2025-12-31","assignedUser":"<USER_ID>"}'

# List tasks completed=false, first 20
curl 'http://localhost:3000/api/tasks?where={"completed":false}&limit=20'
```