Fullstack Home Assignment - Task Manager App

Time Allocation: 4 hours
Stack: React + Express.js + Node.js

Overview

Create a simple Task Manager application where users can create, view, update, and delete tasks. The app should have a React frontend and an Express.js backend with in-memory data storage.

Important Guidelines

Code Authenticity

- All code must be written by you personally
- Use of AI tools (ChatGPT, GitHub Copilot, etc.) or LLMs to generate code is not permitted
- Points will be deducted based on suspicion of AI-generated code
- You may reference documentation and tutorials, but the implementation should be your own

Repository Requirements

- Create a public GitHub repository named: `firstname_lastname_helfy_task` (e.g., `avi_cohen_helfy_task`)
- Include a proper `.gitignore` file to exclude `node_modules` directories
- Repository should contain only source code and necessary configuration files
- Ensure the repository is accessible and properly organized

Requirements

Backend (Express.js/Node.js) - ~90 minutes

Create a REST API with the following endpoints:

Task Model

```javascript
```

```
{
  id: number,
  title: string,
  description: string,
  completed: boolean,
  createdAt: Date,
  priority: 'low' | 'medium' | 'high'
}
```

Required Endpoints

- `GET /api/tasks` - Get all tasks
- `POST /api/tasks` - Create a new task
- `PUT /api/tasks/:id` - Update a task
- `DELETE /api/tasks/:id` - Delete a task
- `PATCH /api/tasks/:id/toggle` - Toggle task completion status

Technical Requirements

- Use Express.js with proper middleware (cors, express.json)
- Store data in memory (array) - no database required
- Include basic input validation
- Add proper error handling with meaningful HTTP status codes
- Use proper REST conventions
- Backend server must run on port 4000

Frontend (React) - ~120 minutes

Create a React application with the following features:

Components Structure

- **App** - Main container component
- **TaskList** - Display all tasks in an endless carousel **(MANDATORY - this is the most important part of the assignment)**
- **TaskItem** - Individual task display with actions
- **TaskForm** - Form for creating/editing tasks
- **TaskFilter** - Filter tasks by completion status

Required Features

Task Model:

```javascript
{
  id: number,
  title: string,
  description: string,
  completed: boolean,
  createdAt: Date,
  priority: 'low' | 'medium' | 'high'
}
```

1. **Display Tasks**: Show all tasks in an endless carousel format
   - Implement smooth infinite scrolling
   - Display tasks in a continuous loop
   - Ensure smooth transitions between tasks
   - Maintain performance with large task lists
   - **CRITICAL**: This must be a real animated carousel with smooth transitions, NOT pagination or simple list scrolling
2. **Add Task**: Form to create new tasks with title, description, and priority
3. **Edit Task**: Ability to edit existing tasks (inline or modal)
4. **Delete Task**: Remove tasks with confirmation
5. **Toggle Completion**: Mark tasks as completed/incomplete
6. **Filter Tasks**: Filter by All/Completed/Pending
7. **Priority Indication**: Visual indication of task priority (colors/badges)

Technical Requirements

- Use React hooks (useState, useEffect)
- Implement endless carousel functionality using vanilla JavaScript/React (no external carousel libraries)
- Make HTTP requests to your backend API
- Handle loading states and errors gracefully
- Responsive design (mobile-friendly)

- Clean, readable code with proper component structure
- Ensure smooth scrolling performance in the carousel
- Handle edge cases for empty task lists in the carousel

Styling - ~30 minutes

- Use regular CSS only (no CSS frameworks, preprocessors, or CSS-in-JS libraries)
- Clean, modern UI design
- Responsive layout
- Visual feedback for different priority levels
- Hover effects and smooth transitions

Evaluation Criteria

Code Quality (25%)

- Clean, readable code structure
- Proper error handling
- Consistent naming conventions
- Comments where necessary

Functionality (35%)

- All CRUD operations work correctly
- Frontend communicates with backend properly
- Filtering and status toggling work
- Form validation and user feedback

UI/UX (20%)

- Intuitive user interface
- Responsive design
- Visual hierarchy and good styling
- Loading states and error messages

Technical Implementation (20%)

- Proper React patterns and hooks usage
- RESTful API design

- Proper HTTP status codes
- Component architecture

Bonus Points (Optional)

If you finish early, consider adding:

- Search functionality
- Sorting options (by date, priority, title)
- Task due dates
- Drag and drop reordering
- Dark/light theme toggle
- Local storage persistence on frontend

Submission Guidelines

IMPORTANT: Proper implementation of the endless animated carousel is mandatory. Applications without a functioning carousel will not be accepted.

File Structure

```
task-manager/
├── backend/
│   ├── package.json
│   ├── server.js
│   ├── routes/
│   └── middleware/
├── frontend/
│   ├── package.json
│   ├── public/
│   ├── src/
│   │   ├── components/
│   │   ├── services/
│   │   ├── styles/
│   │   └── App.js
├── .gitignore
└── README.md
```

What to Submit

1. **GitHub Repository**: Create a public repository with the name format:

`firstname_lastname_helfy_task` (e.g., `avi_cohen_helfy_task`)

2. Submit the GitHub repository link
3. Complete source code for both frontend and backend in the repository
4. **README.md** in the repository root with:
    - Setup and installation instructions
    - How to run both frontend and backend
    - API documentation
    - Any assumptions or design decisions made
    - Time spent on each part
5. Screenshots of the working application (optional)

Setup Instructions Template

```markdown
# Task Manager App

## Backend Setup
1. cd backend
2. npm install
3. npm start (runs on port 4000)

## Frontend Setup
1. cd frontend
2. npm install
3. npm start (runs on port 3000)

## API Endpoints
- GET /api/tasks
- POST /api/tasks
- PUT /api/tasks/:id
- DELETE /api/tasks/:id
- PATCH /api/tasks/:id/toggle
```

Tips for Success

1. Start with the backend API first, test with Postman/curl
2. Build the frontend incrementally, one feature at a time
3. Focus on core functionality before styling

4. Use browser dev tools for debugging
5. Test the full flow before submitting
6. Don't over-engineer - keep it simple and working

Time Management Suggestions

- Backend API: 90 minutes
- Frontend Core Features: 90 minutes
- Styling & Polish: 30 minutes
- Testing & Debugging: 30 minutes

Good luck! Focus on delivering a working application rather than perfect code.