---

1) Additional features — Frontend (React)

Below are concise components that show: categories/labels, priority + due date, recurring tasks, offline PWA hook, sharing via WebSocket, dark mode toggle, export/import CSV, and voice input.

src/components/QuickCreate.jsx

```
// QuickCreate.jsx — task creation with category, priority, due date, recurring & voice
import React, { useState, useRef } from "react";

export default function QuickCreate({ onCreate }) {
  const [title,setTitle]=useState("");
  const [category,setCategory]=useState("General");
  const [priority,setPriority]=useState("Medium");
  const [due,setDue]=useState("");
  const [recurrence,setRecurrence]=useState("none");
  const recognitionRef = useRef(null);

  const startVoice = () => {
    if(!window.SpeechRecognition && !window.webkitSpeechRecognition) return alert("No speech API");
    const R = window.SpeechRecognition || window.webkitSpeechRecognition;
    recognitionRef.current = new R();
    recognitionRef.current.onresult = e => setTitle(e.results[0][0].transcript);
    recognitionRef.current.start();
  };

  const submit = e => {
    e.preventDefault();
    if(!title.trim()) return;
    onCreate({ title, category, priority, due: due||null, recurrence });
    setTitle(""); setDue(""); setRecurrence("none");
  };
```

```
  return (
    <form onSubmit={submit} className="quick-create">
      <input placeholder="Add task..." value={title} onChange={e=>setTitle(e.target.value)} />
      <select value={category} onChange={e=>setCategory(e.target.value)}>

<option>General</option><option>Work</option><option>Personal</option><option>Shopping</o
ption>
      </select>
      <select value={priority} onChange={e=>setPriority(e.target.value)}>
        <option>High</option><option>Medium</option><option>Low</option>
      </select>
      <input type="date" value={due} onChange={e=>setDue(e.target.value)} />
      <select value={recurrence} onChange={e=>setRecurrence(e.target.value)}>
        <option value="none">No repeat</option><option value="daily">Daily</option><option
value="weekly">Weekly</option>
      </select>
      <button type="button" onClick={startVoice}> 🎤 </button>
      <button type="submit">Add</button>
    </form>
  );
}
```

src/hooks/useOfflineSync.js

```
// useOfflineSync.js — store locally & sync when online (very simple)
import { useEffect } from "react";

export default function useOfflineSync(store, syncFn){
  useEffect(()=>{
    const handleOnline = async () => {
      const pending = JSON.parse(localStorage.getItem("pending_tasks")||"[]");
      if(pending.length){
        try{ await syncFn(pending); localStorage.removeItem("pending_tasks"); }
        catch(e){ console.error("Sync failed",e); }
```

```
    }
  };
  window.addEventListener("online", handleOnline);
  return ()=> window.removeEventListener("online", handleOnline);
 },[syncFn]);
}
```

src/service-worker.js (PWA skeleton)

```
// service-worker.js — very small for caching static assets & falling back to offline
self.addEventListener('install', e => {
 e.waitUntil(caches.open('todo-cache-v1').then(c=>c.addAll(['/','/index.html','/bundle.js'])));
});
self.addEventListener('fetch', e => {
 e.respondWith(caches.match(e.request).then(r=>r || fetch(e.request)));
});
```

Export / Import (CSV) utility

```
// exportTasks()
export function exportCSV(tasks){
 const csv = ["title,category,priority,due,recurrence,done",
...tasks.map(t=>`${JSON.stringify(t.title)},${t.category},${t.priority},${t.due||''},${t.recurrence||''},${t
.done?1:0}`)].join("\n");
 const url = URL.createObjectURL(new Blob([csv], {type:'text/csv'}));
 const a = document.createElement('a'); a.href=url; a.download='tasks.csv'; a.click();
}
```

Realtime sharing: client socket snippet (use Socket.io)

```
// connect and listen
import io from "socket.io-client";
const socket = io(process.env.REACT_APP_API_URL);
socket.on("task-updated", data => {/* update local list */});
// when adding/updating:
socket.emit("update-task", updatedTask);
```

---

2) UI/UX Improvements — React snippets & accessibility

- Inline editing: make TaskItem contentEditable or toggle edit input.
- Drag-and-drop: use react-beautiful-dnd or native HTML5 drag/drop. Below minimal inline-edit + aria.

src/components/TaskItem.jsx

```
// TaskItem.jsx — inline edit + keyboard accessible

import React, { useState } from "react";

export default function TaskItem({task, onUpdate, onDelete}){

 const [editing,setEditing]=useState(false), [text,setText]=useState(task.title);

 return (

  <div role="listitem" tabIndex={0} className="task-item">

   {editing ?

    <input autoFocus value={text} onBlur={()=>{onUpdate({...task,title:text}); setEditing(false);}}
onChange={e=>setText(e.target.value)} />

    :

    <div onDoubleClick={()=>setEditing(true)}>{task.title}</div>

   }

   <button aria-label="mark done"
onClick={()=>onUpdate({...task,done:!task.done})}>{task.done?'Undo':'Done'}</button>

   <button aria-label="delete" onClick={()=>onDelete(task.id)}>Delete</button>

  </div>

 );

}
```

- Dark mode: simple CSS toggle persisted in localStorage.
- Animations: use CSS transitions on .task-item for transform/opacity.

---

3) API Enhancements — Backend (Node + Express + Socket.io + GraphQL snippet)

Install:

```
cd backend

npm init -y

npm i express cors helmet bcrypt jsonwebtoken express-rate-limit socket.io graphql express-graphql
prisma @prisma/client redis ioredis
```

backend/src/server.js

```
// server.js — express + REST + socket.io minimal
const express = require('express');
const http = require('http');
const cors = require('cors');
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const { authMiddleware } = require('./auth');
const taskRoutes = require('./routes/tasks');
const { initRealtime } = require('./realtime');

const app = express();
app.use(helmet(),cors(),express.json());
app.use(rateLimit({windowMs:60*1000,max:120})); // 120 req/min

// versioned API
app.use('/api/v1/tasks', authMiddleware, taskRoutes);

// basic health
app.get('/api/v1/health', (req,res)=>res.json({ok:true}));

const server = http.createServer(app);
initRealtime(server); // socket.io setup
server.listen(process.env.PORT||4000,()=>console.log('api up'));
```

backend/src/auth.js (JWT + bcrypt)

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');

const SECRET = process.env.JWT_SECRET || 'devsecret';

async function hashPwd(password){ return await bcrypt.hash(password,10); }
async function comparePwd(pw,hash){ return await bcrypt.compare(pw,hash); }
```

```
function generate(user){ return jwt.sign({id:user.id,email:user.email,role:user.role||'user'}, SECRET, {expiresIn:'7d'}); }

function authMiddleware(req,res,next){
  const h = req.headers.authorization?.split(' ')[1];
  if(!h) return res.status(401).json({error:'no token'});
  try { req.user = jwt.verify(h, SECRET); next(); } catch(e){ res.status(401).json({error:'invalid'}); }
}

module.exports = { hashPwd, comparePwd, generate, authMiddleware };
```

backend/src/routes/tasks.js (REST endpoints + validation)

```
const express = require('express');
const router = express.Router();
// assume simple in-memory store for brevity; replace with DB calls
let tasks = []; // in prod use DB
const { body, validationResult } = require('express-validator');

router.get('/', (req,res)=> {
  const { page=1, limit=50, q } = req.query;
  let out = tasks;
  if(q) out = out.filter(t=>t.title.includes(q));
  const start=(page-1)*limit; res.json({ data: out.slice(start,start+Number(limit)), total:out.length });
});

router.post('/', [
  body('title').isString().trim().notEmpty(),
  body('priority').optional().isIn(['High','Medium','Low'])
], (req,res)=>{
  const err = validationResult(req); if(!err.isEmpty()) return res.status(400).json({errors:err.array()});
  const t = { id:Date.now().toString(), ...req.body, owner:req.user.id, createdAt: new Date().toISOString()};
```

```
  tasks.push(t);
  // emit realtime (socket.io)
  req.app.get('io')?.emit('task-updated', t);
  res.status(201).json(t);
});

router.put('/:id',(req,res)=>{
  const i = tasks.findIndex(x=>x.id===req.params.id); if(i<0) return res.status(404).end();
  tasks[i] = {...tasks[i], ...req.body};
  req.app.get('io')?.emit('task-updated', tasks[i]);
  res.json(tasks[i]);
});

router.delete('/:id',(req,res)=>{
  tasks = tasks.filter(x=>x.id!==req.params.id);
  req.app.get('io')?.emit('task-deleted', {id:req.params.id});
  res.status(204).end();
});

module.exports = router;
```

backend/src/realtime.js

```
// realtime.js — socket.io server
const { Server } = require('socket.io');

function initRealtime(httpServer){
  const io = new Server(httpServer, { cors: { origin: '*' }});
  io.on('connection', socket=>{
    socket.on('update-task', data => io.emit('task-updated', data));
    socket.on('disconnect', ()=>{});
  });
  // make io available to express handlers
```

```
  httpServer.app?.set?.('io', io);
  return io;
}

module.exports = { initRealtime };
```

Notes: replace in-memory store with PostgreSQL via Prisma/Sequelize. Add GraphQL by mounting express-graphql and defining a schema for flexible queries.

---

4) Performance & Security Checks — code & config

Caching (Redis) example — backend/src/cache.js

```
const Redis = require('ioredis');
const redis = new Redis(process.env.REDIS_URL || 'redis://localhost:6379');

async function cacheGet(key){ return JSON.parse(await redis.get(key)); }
async function cacheSet(key,val,ttl=60){ await redis.set(key, JSON.stringify(val), 'EX', ttl); }

module.exports = { redis, cacheGet, cacheSet };
```

Use cacheGet/cacheSet inside GET endpoints (e.g., list of tasks per user).

Secure headers & rate limiting — already added helmet and express-rate-limit in server.js.

Input validation — used express-validator in tasks route.

Password hashing — bcrypt in auth.js.

HTTPS & CORS — configure reverse proxy (Nginx) or let Netlify/Vercel enforce HTTPS. Express: app.set('trust proxy',1) when behind proxies.

---

5) Testing of Enhancements — unit, integration, E2E examples

Unit test example (Jest) — backend/tests/tasks.unit.test.js

```
// jest test skeleton
const request = require('supertest');
const app = require('../src/server'); // make sure server exports app in test mode

describe('tasks API', ()=>{
  it('creates and fetches a task', async ()=>{
```

```
  // assuming test tokens or bypass auth in test env
  const create = await request(app).post('/api/v1/tasks').set('Authorization','Bearer testtoken').send({title:'test',priority:'Low'});
  expect(create.statusCode).toBe(201);
  const list = await request(app).get('/api/v1/tasks').set('Authorization','Bearer testtoken');
  expect(list.body.total).toBeGreaterThan(0);
 });
});
```

Integration (supertest) shown above.

E2E (Cypress) — cypress/integration/add_task.spec.js

```
describe('Add task flow', ()=>{
 it('user can add a task', ()=>{
  cy.visit('/');
  cy.get('input[placeholder="Add task..."]').type('Buy milk');
  cy.get('button').contains('Add').click();
  cy.contains('Buy milk').should('exist');
 });
});
```

Load testing (k6) example script k6_script.js

```
import http from 'k6/http';
import { check } from 'k6';
export default function(){
 let r = http.get(`${__ENV.API_URL}/api/v1/tasks`);
 check(r, {'status 200': r=>r.status===200});
}
```

Security scan: run OWASP ZAP or use npm audit / Snyk in CI.

---

6) Deployment — Netlify / Vercel / Docker + Cloud

Frontend: Netlify (React)

- Build command: npm run build
- Publish directory: frontend/build

- Netlify auto deploys from GitHub. Add netlify.toml for redirects or functions:

```
[build]
  publish = "frontend/build"
  command = "cd frontend && npm ci && npm run build"
```

Frontend: Vercel

- Vercel auto detects React. For Next.js, use Vercel's serverless functions for API routes. Set environment variables in Vercel dashboard.

Backend: Dockerfile (simple)

Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY backend/package*.json ./
RUN npm ci --production
COPY backend/ .
EXPOSE 4000
CMD ["node","src/server.js"]
```

docker-compose.yml (app + redis + postgres)

```
version: '3.8'
services:
  api:
    build: .
    ports: ['4000:4000']
    environment:
      - DATABASE_URL=postgres://postgres:pass@db:5432/todo
      - REDIS_URL=redis://redis:6379
    depends_on: ['db','redis']
  db:
    image: postgres:15
    restart: always
    environment: POSTGRES_PASSWORD: pass
    volumes: ['db-data:/var/lib/postgresql/data']
```

```
  redis:
    image: redis:7
volumes:
  db-data:
```

CI/CD: GitHub Actions (basic)

.github/workflows/ci.yml

```
name: CI
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    services:
      postgres: { image: postgres:15, env: POSTGRES_PASSWORD: pass, ports: ['5432:5432'] }
      redis: { image: redis:7, ports: ['6379:6379'] }
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: node-version: 18
      - run: cd backend && npm ci && npm test
      - run: cd frontend && npm ci && npm test
  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
      - uses: actions/checkout@v4
      - name: Deploy to Docker Hub (example)
        run: echo "deploy step..."
```