## Planification Notes Archive

## 🔍 Requirement Gathering (with Redis)

### 🎯 Project Goal:

Build a simple application that allows users to add short notes and temporarily store them in a fast in-memory storage (Redis), with the ability to view all saved notes.

### 📌 Functional Requirements:

* The user can type a note in an input field.
* When clicking the "Add" button, the note is sent to the backend.
* The backend stores the note in Redis (instead of an array).
* The user can view all notes stored in Redis.
* The storage is temporary (cache), not persistent like a database.
* Communication between React frontend and Node/Express backend is done via REST API.

### 🚫 Non-Functional Requirements:

* ❌ No user authentication required.
* ❌ No note editing or deleting functionality.
* ✅ Redis is used for fast, flexible caching with automatic expiration (TTL).
* ✅ High performance thanks to Redis speed.
* ✅ The UI is simple and easy to use.

### 🛠️ Constraints:

* Redis must be running locally or on a server.
* Data does not persist after a server restart unless Redis persistence is configured (optional).
* Note: Redis is used as a cache, not for permanent data storage.

## 🧱 System Design

### 🏗️ Architecture Overview:

* **Frontend:**
  + React app responsible for the user interface (input field + notes list display).
  + Communicates with the backend via REST API.
* **Backend:**
  + Node.js with Express server.
  + API endpoints to add and retrieve notes.
  + Stores notes in Redis instead of an in-memory array.
* **Storage:**
  + Redis as an in-memory datastore.
  + Temporary storage with TTL (time-to-live) for each note if needed.

### 🔌 API Endpoints:

| Endpoint | Method | Description |
| --- | --- | --- |
| /notes | GET | Fetch all notes from Redis |
| /notes | POST | Add a new note to Redis |

### 🧩 Components Details:

#### Frontend (React):

* **Input Component:**  
  Text input for entering a note and a button to submit it.
* **Notes List Component:**  
  Displays notes fetched from the backend.
* **Axios / fetch:**  
  Used to send and receive data to/from the backend.

#### Backend (Express + Redis):

* **Express Server:**  
  Handles GET and POST requests on /notes.
* **Redis Client:**  
  Connects to Redis server.
  + On POST: adds the note to a Redis list (e.g., using LPUSH or RPUSH).
  + On GET: retrieves all notes using LRANGE.
* **Data Structure in Redis:**  
  Notes stored in a list with a specific key (e.g., notes\_list).
* **TTL (optional):**  
  Can set expiration time for each note or the whole list if desired.

### 🛠️ Deployment Considerations:

* Redis must be installed and running locally or on an external server.
* Configure CORS in Express to allow React frontend communication.
* Local environment runs frontend and backend separately (React dev server and Node server).

## 📦 Work Breakdown Structure (WBS)

### 1. Environment Setup

* Install Node.js and npm
* Initialize React project (create-react-app or Vite)
* Initialize Node.js with Express project
* Install and configure Redis locally or on a server

### 2. Frontend Development

* Design a simple main page
* Create Input component for adding notes
* Create Notes List component to display notes
* Connect frontend to backend API using fetch or axios

### 3. Backend Development

* Setup Express server
* Create REST API endpoints:
  + GET /notes to retrieve notes
  + POST /notes to add a new note
* Configure Redis connection
* Implement logic to store and retrieve notes from Redis

### 4. Unit Testing

* Test React components (Input and Notes List)
* Test backend API endpoints
* Test Redis connectivity and operations

### 5. Integration Testing

* Test interaction between frontend and backend
* Verify data storage and retrieval from Redis works correctly

### 6. Deployment Preparation

* Prepare production build of React app
* Setup server environment to host backend and Redis (local or cloud)
* Configure CORS and security settings

### 7. Deployment & Monitoring

* Deploy the application to production environment
* Monitor app and Redis performance
* Prepare incident management and error handling plan

## 4. ⏱ Estimation

| Task | Estimated Time | Notes |
| --- | --- | --- |
| **1. Environment Setup** | 2 hours | Installing tools, initializing projects, Redis setup |
| **2. Frontend Development** | 6 hours | Designing UI, creating Input and Notes List components, API integration |
| **3. Backend Development** | 6 hours | Setting up Express, building API, connecting to Redis, storing and retrieving data |
| **4. Unit Testing** | 3 hours | Testing React components, API endpoints, and Redis connectivity |
| **5. Integration Testing** | 2 hours | Testing interaction between frontend and backend |
| **6. Deployment Preparation** | 3 hours | Preparing production build, server setup, CORS and security configuration |
| **7. Deployment & Monitoring** | 2 hours | Deploying the app, monitoring performance, error handling |

### ⏳ ****Total Estimated Time:**** Around 24 hours (approximately 3 working days)

### Notes:

* Estimates depend on your experience and project complexity.
* Additional time might be needed for debugging or learning during development.
* Tasks can be divided among a team or prioritized differently.

## 5. 🗓 Scheduling & Timeline

| Week / Day | Task | Description |
| --- | --- | --- |
| **Day 1** | Environment Setup | Install Node.js, React, Redis; initialize projects |
| **Day 2 - Morning** | Frontend Development Start | Build Input component and Notes List UI |
| **Day 2 - Afternoon** | Frontend Development Continue | Connect frontend with backend API |
| **Day 3 - Morning** | Backend Development Start | Setup Express server, create API endpoints |
| **Day 3 - Afternoon** | Backend Development Continue | Integrate Redis for storing and retrieving notes |
| **Day 4** | Unit Testing | Test frontend components, backend API, Redis |
| **Day 5** | Integration Testing | Test frontend-backend interaction |
| **Day 6** | Deployment Preparation | Production build, configure server, CORS, security |
| **Day 7** | Deployment & Monitoring | Deploy application, monitor performance, fix bugs |

### Summary:

* **Total Duration:** 7 days (1 week)
* **Daily Focus:** Each day focuses on specific key parts for better productivity and progress tracking
* **Flexibility:** Can be adjusted based on team size and availability

## 6. 🧪 Testing Plan

### 1. Unit Testing

* **Frontend:**
  + Test Input component functionality (adding notes)
  + Test Notes List component (displaying notes correctly)
* **Backend:**
  + Test API endpoints:
    - POST /notes (adding note)
    - GET /notes (retrieving notes)
* **Redis:**
  + Test connection to Redis server
  + Test storing and retrieving notes from Redis

### 2. Integration Testing

* Test full flow from frontend input to backend storage and retrieval
* Verify API calls are working correctly between React frontend and Express backend
* Ensure data consistency between frontend display and Redis storage

### 3. Performance Testing

* Test Redis response times for storing and fetching notes
* Check app responsiveness with multiple notes

### 4. Error Handling Testing

* Simulate Redis connection failure and check system behavior
* Test API error responses for invalid requests
* Validate frontend handles errors gracefully (e.g., show error messages)

### 5. User Acceptance Testing (UAT)

* Verify that users can add notes and see them immediately
* Confirm no login, edit, or delete features are available as per requirements
* Check UI usability and responsiveness

### Tools Suggested:

* **Frontend:** Jest, React Testing Library
* **Backend:** Jest, Supertest
* **Redis:** Mock Redis or integration tests with local Redis instance

## 7. 🚀 Deployment & Monitoring Plan

### Deployment

* **Prepare Production Build:**
  + Build React frontend for production (e.g., npm run build)
  + Setup Node.js/Express backend for production environment
* **Server Setup:**
  + Choose hosting platform (e.g., AWS EC2, DigitalOcean, Heroku)
  + Install and configure Redis server (locally or managed service)
  + Deploy backend and frontend apps on the server
* **Configuration:**
  + Configure environment variables (API URLs, Redis connection)
  + Setup CORS policies for secure communication
  + Enable SSL for HTTPS security
* **Automation:**
  + Use CI/CD pipelines for automated deployment (optional)
  + Dockerize app for easier deployment and scaling (optional)

### Monitoring

* **Logging:**
  + Implement logging on backend (errors, API calls)
  + Store logs in files or use logging services (e.g., Loggly, ELK stack)
* **Performance Monitoring:**
  + Monitor server resource usage (CPU, memory)
  + Track Redis performance metrics (cache hits, misses)
* **Error Monitoring:**
  + Use tools like Sentry or New Relic to catch runtime errors
  + Setup alerts for critical failures
* **Health Checks:**
  + Setup endpoint health checks to verify app availability
  + Use uptime monitoring tools (e.g., UptimeRobot)

### Backup & Recovery (Optional)

* Setup Redis persistence if you want to keep cache after restarts
* Regular backups of Redis data if needed