Thru My Lens - Technical Documentation

Overview

Website: https://rosy.shitchell.com/photography/ **Type:** Personal Photography Portfolio **Created:** December 2024

Technology Stack

Frontend

Technology	Version	Purpose
HTML5	-	Page structure and semantic markup
CSS3	-	Styling, animations, responsive design
Vanilla JavaScript	ES6+	Interactivity, API calls, DOM manipulation
Google Fonts	-	Typography (Cormorant Garamond, Montserrat)

No frameworks used - Pure HTML, CSS, and JavaScript for simplicity and performance.

Backend

Technology	Version	Purpose
Node.js	18+	Server runtime for API
HTTP module	Built-in	Web server (no Express needed)
FS module	Built-in	File system operations
Crypto module	Built-in	Generating unique filenames

Infrastructure

Technology	Version	Purpose
Nginx	1.24+	Web server, reverse proxy, SSL termination
Docker	24+	Container runtime for API
Docker Compose	2+	Container orchestration
Let’s Encrypt	-	SSL/TLS certificates
Ubuntu Linux	24.04	Server operating system

Data Storage

Technology	Purpose
File System	Photo storage (JPEG, PNG, WebP, GIF)
JSON	Metadata storage (titles, sort order, upload dates)

No database used - Simple JSON file for metadata keeps things lightweight.

Development Tools

Code Editing

- **Claude Code** - AI-assisted coding and terminal operations

Version Control

- **Git** - Source control
- **GitHub** - Repository hosting, deployment triggers

Testing

- **Browser DevTools** - Chrome/Firefox developer tools for debugging
- **curl** - API endpoint testing
- **Docker logs** - Server-side debugging

Deployment

- **GitHub Actions** - Automated deployment on push to main
- **rsync** - File synchronization to server

File Structure

```
/var/www/rosy.shitchell.com/srv/
├── web/
│   └── photography/
│       ├── index.html              # Main gallery (all frontend
code)
│       ├── upload.html             # Upload interface
│       └── uploads/
│           ├── landscapes/         # Nature photos
│           │   └── *.jpg, *.png
│           ├── historic/           # City photos
│           │   └── *.jpg, *.png
│           └── metadata.json       # Photo titles, order, dates
│
└── upload-api/
    ├── server.js                   # Node.js API server
    ├── Dockerfile                  # Container build instructions
    └── docker-compose.yml          # Container configuration
```

Frontend Architecture

Single-File Approach

All frontend code lives in `index.html`: - **CSS** in `<style>` tags (~600 lines) - **HTML** structure (~150 lines) - **JavaScript** in `<script>` tags (~500 lines)

Why single file? - Simple to edit and deploy - No build process needed - Easy to understand entire application - Appropriate for small project scope

CSS Features Used

Feature	Usage
CSS Variables	Color palette (`:root { --bg, --ink, etc. }`)
Flexbox	Navigation, modal buttons, menu items
CSS Grid	Photo gallery layout
clamp()	Responsive typography
backdrop-filter	Frosted glass effect on navigation
Transitions	Hover effects, animations
Media Queries	Mobile responsive breakpoints
aspect-ratio	Square photo frames

JavaScript Features Used

Feature	Usage
async/await	API calls
Fetch API	HTTP requests to backend
FormData	File uploads
DOM manipulation	Dynamic gallery rendering
Event listeners	Click, drag, keyboard handlers
Drag and Drop API	Photo reordering
FileReader API	Image preview before upload
classList	Toggle CSS classes
Template literals	HTML string generation
Arrow functions	Callbacks and event handlers
Destructuring	API response handling

No External Libraries

The frontend uses zero external JavaScript libraries: - No jQuery - No React/Vue/Angular - No Lodash/Underscore - No Moment.js

Everything is built with native browser APIs.

Backend Architecture

API Server Design

Simple Node.js HTTP server without frameworks:

```
const http = require('http');
const fs = require('fs');
const path = require('path');
const crypto = require('crypto');
```

```
const server = http.createServer((req, res) => {
    // Route handling with if/else
    if (req.method === 'GET' && req.url === '/photos') {
        // List photos
    }
    if (req.method === 'POST' && req.url === '/upload') {
        // Handle upload
    }
    // etc.
});
```

Why no Express.js? - Only 6 endpoints, no complex routing needed - Fewer dependencies = fewer security updates - Educational value in understanding raw HTTP - Smaller container image

API Endpoints

Method	Endpoint	Request Body	Response
GET	`/photos`	-	`{ landscapes: [...], historic: [...] }`
POST	`/upload`	FormData (photo, title, category, password)	`{ success, filename, path }`
POST	`/edit`	JSON (filename, category, title, password)	`{ success }`
POST	`/replace`	FormData (photo, filename, category, password)	`{ success, newFilename, path }`
POST	`/delete`	JSON (filename, category, password)	`{ success }`
POST	`/reorder`	JSON (category, order[], password)	`{ success }`

File Upload Handling

Custom multipart form parser (no multer or busboy):

```
function parseMultipart(buffer, boundary) {
    // Parse boundary-separated parts
    // Extract filename, content-type, data
    // Return array of parts
}
```

Security measures: - File extension whitelist (`.jpg`, `.jpeg`, `.png`, `.gif`, `.webp`) - Max file size limit (20MB) - Filename sanitization (remove special characters) - Unique filename generation (timestamp + random hash) - Password authentication on all write operations

Metadata Storage

`metadata.json` structure:

```
{
  "photo_1234567890_abc123.jpg": {
    "title": "Sunset at the Beach",
    "category": "landscapes",
    "sortOrder": 0,
    "uploaded": "2024-12-08T12:00:00.000Z"
  },
```

```
    "photo_0987654321_def456.jpg": {
      "title": "City Lights",
      "category": "historic",
      "sortOrder": 1,
      "uploaded": "2024-12-08T13:00:00.000Z"
    }
  }
```

Infrastructure Setup

Nginx Configuration

```
server {
    server_name rosy.shitchell.com;
    root /var/www/rosy.shitchell.com/srv/web;

    # Static files
    location /photography/ {
        try_files $uri $uri/ =404;
    }

    # API proxy
    location /upload-api/ {
        proxy_pass http://127.0.0.1:8083/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        client_max_body_size 25M;
    }

    # SSL (managed by Certbot)
    listen 443 ssl;
    ssl_certificate
/etc/letsencrypt/live/rosy.shitchell.com/fullchain.pem;
    ssl_certificate_key
/etc/letsencrypt/live/rosy.shitchell.com/privkey.pem;
}
```

Docker Configuration

Dockerfile:

```
FROM node:18-alpine
WORKDIR /app
COPY server.js .
CMD ["node", "server.js"]
```

docker-compose.yml:

```
services:
  upload-api:
    build: .
    container_name: rosy-upload
    restart: unless-stopped
    ports:
      - "127.0.0.1:8083:3000"
    volumes:
      - /var/www/.../uploads:/uploads
      - /var/www/.../server.js:/app/server.js:ro
    environment:
      - UPLOAD_PASSWORD=<secret>
```

Key decisions: - `127.0.0.1:8083` - Only accessible locally (Nginx proxies external requests) - Volume mount for `server.js` - Code changes without rebuilding container - Volume mount for `uploads` - Photos persist outside container - `restart: unless-stopped` - Auto-restart on crash or server reboot

---

Security Measures

Authentication

- Password stored in environment variable (not in code)
- Required for all write operations (upload, edit, delete, replace, reorder)
- Sent with each request (no session tokens)

File Security

- Whitelist of allowed extensions
- Filenames sanitized to alphanumeric + underscore
- Original filenames never used directly
- Files stored outside web root with controlled access

Network Security

- HTTPS only (SSL/TLS via Let's Encrypt)
- API only accessible through Nginx proxy
- Docker container isolation
- CORS restricted to `https://rosy.shitchell.com`

Input Validation

- File size limits enforced
- Category whitelist (only `landscapes` or `historic`)
- JSON parsing in try/catch blocks

Performance Optimizations

Frontend

- `loading="lazy"` on images (browser-native lazy loading)
- CSS transitions (GPU-accelerated)
- No external JavaScript libraries to download
- Single HTTP request for page (inline CSS/JS)

Backend

- Synchronous file operations (appropriate for low traffic)
- JSON metadata cached in memory (read on each request, but file is small)
- Static files served directly by Nginx (not through Node.js)

Images

- `object-fit: cover` for consistent display without distortion
- Original resolution preserved (no server-side resizing)
- Browser caching via Nginx headers

Browser Compatibility

Supported Browsers

- Chrome 80+
- Firefox 75+
- Safari 13+
- Edge 80+

Features That Require Modern Browsers

- CSS Grid
- CSS `clamp()`
- CSS `aspect-ratio`
- `backdrop-filter` (frosted glass effect)
- Drag and Drop API
- Fetch API
- async/await

Graceful Degradation

- Navigation still works without JavaScript (anchor links)
- Photos display even if JS fails (server-rendered paths)
- Blur effect degrades to solid background on older browsers

Deployment Process

Manual Deployment

1. Edit files in `/var/www/rosy.shitchell.com/srv/web/photography/`
2. Changes are live immediately (static files)
3. For API changes: `docker restart rosy-upload`

Git-Based Deployment

1. Edit files locally or in `~/code/git/rosy-web/`
2. Commit and push to main branch
3. GitHub Actions runs rsync to server
4. API container restarts if server.js changed

Rollback

- Git revert for code changes
- Photos not in Git (manual backup needed)
- metadata.json can be restored from backup

Monitoring & Debugging

Logs

```
# Nginx access logs
tail -f /var/log/nginx/access.log

# Nginx error logs
tail -f /var/log/nginx/error.log

# API container logs
```

```
docker logs -f rosy-upload
```

Health Checks

```
# Check if API is running
curl https://rosy.shitchell.com/upload-api/photos

# Check container status
docker ps | grep rosy-upload
```

Common Issues

Problem	Solution
Photos not loading	Check file permissions in uploads folder
API returns 404	Restart container: `docker restart rosy-upload`
Upload fails	Check Nginx `client_max_body_size` setting
CORS errors	Verify origin in server.js matches domain

Local Development

Prerequisites

- Node.js 18+
- Docker (optional, for containerized testing)

Running Locally

```
# Start API server
cd /var/www/rosy.shitchell.com/srv/upload-api
node server.js

# Serve frontend (any static server)
cd /var/www/rosy.shitchell.com/srv/web/photography
python3 -m http.server 8000
```

Testing API

```
# List photos
curl http://localhost:3000/photos

# Upload (requires form data)
curl -X POST http://localhost:3000/upload \
  -F "photo=@test.jpg" \
  -F "title=Test" \
  -F "category=landscapes" \
  -F "password=your-password"
```

Summary

Layer	Technology	Why Chosen
Frontend	HTML/CSS/JS	Simple, no build step, easy to learn
Backend	Node.js	JavaScript everywhere, lightweight
Server	Nginx	Fast, reliable, good SSL support
Container	Docker	Isolation, reproducibility
Storage	Files + JSON	Simple, no database overhead

SSL	Let’s Encrypt	Free, automated renewal
Hosting	Ubuntu VPS	Full control, cost-effective

Total external dependencies: 2 (Google Fonts for typography)

Document created: December 2024 Website: https://rosy.shitchell.com/photography/