Sree Andhra Valmiki Portal
Knowledge Transfer (KT) Document

Frontend + Backend (React + Node/Express + MongoDB)

Prepared for: Cloud Manager / Deployment & Support Team

Date: 14 Jan 2026

Table of Contents

1. Project Overview

Sree Andhra Valmiki Portal is an education + devotional content website. It supports user login, admin content management, and hosting/serving media such as YouTube video links, images, PDFs, audio files, banners, and gallery albums.

Primary goals:

- Public website for devotional/educational content (Home, Articles, Gallery, Videos, About).
- Secure user login with OTP verification and JWT-based sessions.
- Admin dashboard to upload/manage banners, PDFs, images, audio, videos, and notifications.
- MongoDB used to store metadata; uploaded files stored in server filesystem (or object storage in production).

2. Repository Structure

Recommended monorepo layout (as merged):

Path	Purpose
media-upload-api/	Backend Node/Express API (MongoDB). Also contains frontend build (optional).
media-upload-api/models/	Mongoose models (User, OtpSession, Notification, etc.).
media-upload-api/uploads/	Uploaded files (banners, images, pdfs, audio, videos, gallery folders).
media-upload-api/frontend/	React frontend source (if included).
media-upload-api/frontend/dist/	Production build output (if built and served by backend).
media-upload-api/server.js	Express app entrypoint (routes + static serving + middleware).
media-upload-api/authentication.js	Auth routes (OTP, password login, admin login, JWT).
media-upload-api/*Routes.js	Feature routes (content, gallery, notifications, uploads, videos, audio, etc.).

Important Git rules:

- Do NOT commit .env (secrets). Commit only .env.example.

- Do NOT commit node_modules. Commit package.json and package-lock.json.
- Ignore __MACOSX/ and other OS junk files.
- Uploads should usually be excluded from Git (use object storage in production).

3. Environment & Configuration

Backend configuration is controlled via environment variables in media-upload-api/.env.

Key variables (example):

ENV Key	Used For	Notes
PORT	Backend port	Default 5000
MONGO_URI	MongoDB connection string	Atlas or self-hosted MongoDB
JWT_SECRET	JWT signing secret	Must be strong in production
JWT_EXPIRES_IN	Token expiry	Example: 7d
GMAIL_USER	Email OTP sender	Use App Password
GMAIL_APP_PASSWORD	Email OTP sender password	App Password only
TWILIO_ACCOUNT_SID	SMS OTP (optional)	If SMS OTP enabled
TWILIO_AUTH_TOKEN	SMS OTP (optional)	If SMS OTP enabled
TWILIO_FROM_NUMBER	SMS OTP (optional)	If SMS OTP enabled
STATIC_ADMIN_EMAIL	Static admin login (optional)	For quick admin access
STATIC_ADMIN_PASSWORD	Static admin login (optional)	Store only in env

Security notes:

- Never commit secrets (JWT_SECRET, DB URIs, SMTP/Twilio credentials).
- Enable rate limiting on OTP endpoints (already present via express-rate-limit in authentication.js).
- Use HTTPS in production and secure cookies/storage as applicable.

4. Backend Overview (Node/Express)

Backend is an Express server that exposes REST APIs under /api and connects to MongoDB using Mongoose.

Common modules:

- server.js: Express app setup, middleware (CORS, JSON parsing), route mounting, static frontend serving (optional).
- models/: Mongoose schemas for persisted objects (User, OtpSession, Notification, etc.).
- authentication.js: OTP + password login + admin login + JWT issue/verify.
- notificationRoutes.js: CRUD for admin notifications shown in UI.
- galleryRoutes.js: Folder-based gallery CRUD and image upload management.
- contentRoutes.js: CRUD for PDFs and other content modules (articles/videos/audio), depending on implementation.
- upload routes: multipart upload for banners/images/audio/videos/pdfs stored under uploads/.

5. Authentication & Login APIs (backend authentication.js)

Login is implemented with OTP verification (email/SMS) and JWT. Frontend uses these endpoints (based on current integration in src/pages/Login.jsx):

API	Method	Path	Purpose	Auth	Request/ Response (high level)
Request Login OTP	POST	/api/auth/ login/ request-otp	Send OTP to email/phone	Public	Req: { email | phone }. Res: { success, message, sessionId? }
Verify Login OTP	POST	/api/auth/ login/verify-otp	Verify OTP and allow password login	Public	Req: { email| phone, otp }. Res: { success, token, user }
Password Login (User)	POST	/api/auth/ login/ password	Login using password (after OTP verified once)	Public	Req: { email, password }. Res: { success, token, user }
Signup (User)	POST	/api/auth/ signup	Create user account	Public	Req: { name, email, phone, password }. Res: { success,

					user }
Get Current User	GET	/api/auth/me	Fetch logged-in user details	Bearer JWT	Res: { success, user }
Admin Login	POST	/api/auth/admin/login	Admin password login (static env credentials)	Public	Req: { email, password }. Res: { success, token, role: 'admin' }

JWT usage:

- On successful login, backend issues JWT signed with JWT_SECRET.
- Frontend stores token (commonly localStorage) and attaches it in Authorization: Bearer <token> for protected APIs.
- Protected routes should validate JWT and set req.user.

6. Content & Media APIs

The portal stores media in two parts: (1) metadata in MongoDB and (2) actual files in uploads/ folder (or object storage in production).

Known API base URLs used in frontend:

- API base: http://localhost:5000/api
- Content APIs: /api/content/...
- PDF API (known): /api/content/pdfs
- Upload banner: /api/upload/banner
- Videos API (known): /api/videos
- Audio API (known): /api/audio (or similar depending on code)

Module	Typical Endpoints	Stored In DB	Stored As File	Notes
Banners	POST /api/upload/banner, GET /api/content/banners	Banner metadata	uploads/banners/	Used on home page slideshow
PDFs	POST/GET/DELETE	PDF metadata	uploads/pdfs/	Displayed in portal +

	/api/content/pdfs			downloadable
Images	POST /api/upload/image, GET /api/content/images	Image metadata	uploads/images/	Used for posts/thumbnails
Audio	POST /api/audio, GET /api/audio	Track metadata	uploads/audio/	Bhajans / pravachan audio
Videos	POST/GET /api/videos	Video records	N/A (YouTube links) or uploads/videos/	Prefer YouTube URL to reduce storage

Note: Exact paths for some media endpoints can vary based on your route files. Confirm via server.js route mounting and *Routes.js files.

7. Notifications APIs

Implemented in notificationRoutes.js with MongoDB model Notification.

Method	Path	Purpose	Body	Notes
POST	/api/notifications	Create notification	{ text }	Creates {text, isRead:false, createdAt}
GET	/api/notifications	List notifications	-	Sorted by createdAt desc
PUT	/api/notifications/:id	Update notification	{ text?, isRead? }	Mark read/unread or edit text
DELETE	/api/notifications/:id	Delete notification	-	Admin-only action (recommend auth)

8. Gallery APIs

Folder based gallery: create folders, upload images into folders, update captions, delete.

Known endpoints list (as integrated earlier):

Action	Method	Endpoint	Notes
List folders	GET	/api/gallery/folders	Returns list of folder objects
Create folder	POST	/api/gallery/folders (FormData)	Fields: title + optional thumbnail
Update folder	PUT	/api/gallery/folders/:id	Edit title/description
Update thumbnail	POST	/api/gallery/folders/:id/thumbnail (FormData)	Uploads and sets new thumbnail
Delete folder	DELETE	/api/gallery/folders/:id	Also delete folder files if implemented
List images in folder	GET	/api/gallery/folders/:id/images	Returns list of images
Upload images	POST	/api/gallery/folders/:id/images (FormData)	Multi-file upload
Update caption	PUT	/api/gallery/images/:imageId	Edit image caption
Delete image	DELETE	/api/gallery/images/:imageId	Remove image and file

9. Frontend Overview (React)

Frontend is a React single-page application with React Router.

Core pages (as seen in App.jsx):

- Login (src/pages/Login.jsx) - OTP and password login for users + admin login.
- Splash (src/pages/Splash.jsx) - post-login landing or loading.
- User pages under MainLayout: Home, Articles, Gallery, Videos, About.
- Admin pages: HomeAdmin, GalleryAdmin (and other admin modules).

API integration points:

- src/utils/authApi.js (or similar): wrappers for signupUser, requestLoginOtp, verifyLoginOtp, getMe.
- Admin pages call APIs: /api/notifications, /api/gallery/..., /api/content/pdfs, /api/videos, /api/upload/banner, etc.
- Token is used in request headers for protected endpoints.

10. Frontend Auth Flow (src/pages/Login.jsx + utils/authApi.js)

Two login modes are supported:

- User login: OTP request -> OTP verify -> (optional) password login.
- Admin login: direct password login using /api/auth/admin/login.

Frontend constants (as currently configured):

Constant	Value / Meaning
API_BASE_URL	http://localhost:5000/api
PASSWORD_LOGIN_URL	/auth/login/password
ADMIN_PASSWORD_LOGIN_URL	/auth/admin/login
USER_HOME	/splash
ADMIN_HOME	/admin (or your actual admin route)

Recommended token storage approach:

- Store JWT token in localStorage (simple) OR httpOnly cookie (more secure).
- Attach token to API requests: Authorization: Bearer <token>.
- On logout: remove token and redirect to /login.

11. Admin Panel Pages & API Mapping

Admin modules typically include content CRUD and uploads. Example mapping:

Admin Page / Component	Primary APIs	What it Manages
HomeAdmin.jsx	/api/notifications, /api/upload/banner, /api/videos, /api/audio, /api/content/pdfs	Home marquee, banners, quick content controls

GalleryAdmin.jsx	/api/gallery/folders, /api/gallery/folders/:id/images	Gallery folders + image uploads + captions
Home.jsx / public pages	GET content APIs	Public viewing of content

12. Deployment Notes (Server + Storage)

Deployment requires:

- Node.js server (Express) running behind reverse proxy (Nginx).
- MongoDB (Atlas or managed DB).
- Storage for uploads: local disk for dev; object storage (S3 / Cloudflare R2) recommended for production.
- CDN for static media to improve performance (optional).

If serving React build from backend:

- Build frontend: npm run build (in frontend folder).
- Serve frontend/dist as static in server.js.
- All non-API routes should fallback to index.html (React Router).

13. Testing / Use Cases & Test Cases (Smoke Checklist)

Use these test cases during KT or after deployment.

13.1 Authentication

- Request OTP with valid email/phone -> should receive OTP (or see success response).
- Verify OTP with correct code -> should receive JWT token.
- Verify OTP with wrong/expired code -> should get error.
- User password login -> token returned; /api/auth/me returns user.
- Admin login -> token returned; admin routes accessible.

13.2 Content & Media

- Upload banner image -> appears in Home UI.
- Upload PDF -> list PDFs and download works.
- Create video entry (YouTube link) -> video page renders.
- Upload audio -> audio list + playback works.

13.3 Gallery

- Create folder with title -> folder appears.
- Upload multiple images to a folder -> images list shows.
- Update caption -> caption changes.

- Delete image -> removed from UI and storage.

13.4 Notifications

- Create notification -> appears in admin + user header badge.
- Mark read/unread -> count updates.
- Delete notification -> removed from list.

14. Troubleshooting

- Frontend not running: ensure correct Node version; run npm install and npm start in frontend folder.
- CORS issues: confirm backend CORS config allows frontend origin.
- Mongo connection errors: verify MONGO_URI and IP allowlist (Atlas).
- OTP email fails: use Gmail App Password; antivirus SSL inspection can break SMTP - disable or whitelist.
- Large file uploads fail: increase body limits and reverse proxy limits (Nginx client_max_body_size).

15. Appendix: Example Requests

A) Create notification

curl -X POST "http://localhost:5000/api/notifications" -H "Content-Type: application/json" -d '{"text":"New update"}'

B) List gallery folders

curl "http://localhost:5000/api/gallery/folders"

C) Password login (user)

curl -X POST "http://localhost:5000/api/auth/login/password" -H "Content-Type: application/json" -d '{"email":"user@example.com","password":"*********"}'