Backend folder structure (detailed) — Headly

Nice — here's a complete, practical backend folder layout for **Headly** (Express + Mongoose). It's organized for clarity, scalability, and real-world needs (auth, media, webhooks, jobs, tests, observability). I include short descriptions for each folder/file and a few suggested npm scripts & env vars at the end so you can wire it up quickly.

```
backend/
├─ .env.example
├─ .editorconfig
├─ .eslintrc.js
├─ .prettierrc
├─ package.json
├─ README.md
├─ Dockerfile
├─ docker-compose.yml
├─ pm2.ecosystem.config.js
├─ scripts/
│  ├─ seed-admin.js
│  └─ migrate-media.js
├─ public/                  # (optional) public assets served by express
│  └─ uploads/
├─ uploads/                 # server-side temporary uploads (gitignored)
│  └─ tmp/
├─ src/
│  ├─ server.js             # entry: bootstraps app & DB
│  ├─ app.js                # express app, middlewares, routes mount
│  ├─ config/
│  │  ├─ index.js           # central config loader (env + defaults)
│  │  ├─ db.js              # mongoose connection
│  │  └─ cloudinary.js      # cloudinary or s3 client init
│  ├─ api/
│  │  ├─ v1/                # versioned API
│  │  │  ├─ index.js        # router aggregator for v1
│  │  │  ├─ contents/
│  │  │  │  ├─ contents.controller.js
│  │  │  │  ├─ contents.routes.js
│  │  │  │  ├─ contents.service.js
│  │  │  │  └─ contents.validators.js
│  │  │  ├─ media/
│  │  │  │  ├─ media.controller.js
│  │  │  │  ├─ media.routes.js
│  │  │  │  └─ media.service.js
```

```
│   │   ├── auth/
│   │   │   ├── auth.controller.js
│   │   │   ├── auth.routes.js
│   │   │   └── auth.service.js
│   │   ├── users/
│   │   │   ├── users.controller.js
│   │   │   ├── users.routes.js
│   │   │   └── users.service.js
│   │   └── webhooks/
│   │       ├── webhook.controller.js
│   │       └── webhook.routes.js
│   ├── models/
│   │   ├── Content.js
│   │   ├── Media.js
│   │   ├── User.js
│   │   └── AuditLog.js
│   ├── repositories/          # DB access layer (optional but recommended)
│   │   ├── content.repo.js
│   │   └── media.repo.js
│   ├── middlewares/
│   │   ├── auth.middleware.js
│   │   ├── rbac.middleware.js
│   │   ├── error.handler.js
│   │   ├── rateLimit.middleware.js
│   │   └── validate.middleware.js
│   ├── utils/
│   │   ├── logger.js          # winston/pino wrapper
│   │   ├── responses.js       # standardized API responses
│   │   ├── slugify.js
│   │   └── email.js           # sendgrid/mail wrapper (notifications)
│   ├── jobs/                  # background workers / cron jobs
│   │   ├── index.js           # job runner
│   │   ├── publishScheduler.js  # scheduled publish worker
│   │   └── searchIndexer.js     # indexing on publish for Algolia/Elastic
│   ├── integrations/
│   │   ├── algolia.client.js
│   │   ├── analytics.client.js
│   │   └── social.client.js     # twitter/linkedin webhook helpers
│   ├── validators/            # Joi / Yup schemas
│   │   ├── content.validator.js
│   │   └── auth.validator.js
│   └── tests/
│       ├── unit/
│       │   └── content.service.test.js
```

```
│   └── integration/
│       └── contents.routes.test.js
├── .gitignore
└── CHANGELOG.md
```

File / folder purpose (short explainer)

- **.env.example** — Required env vars with descriptions (DB URL, secrets, cloud creds, frontend URL, etc.).
- **Dockerfile / docker-compose.yml** — containerize the app + MongoDB/local services.
- **scripts/** — helper scripts (seed admin user, migrate old media, DB seeds).
- **public/uploads** & **uploads/tmp** — temporary storage for file uploads (should be gitignored; production upload goes to Cloudinary/S3).
- **src/server.js** — entry: require config, connect DB, start HTTP server and job scheduler.
- **src/app.js** — defines express app, global middlewares (helmet, cors, bodyParser), mounts API routers and health checks.
- **src/config/** — centralized config (read process.env, set defaults), DB connect logic, CDN clients.
- **src/api/v1/** — versioned routes, each feature in its own folder with controller/service/routes/validators. Keeps controllers thin and business logic in services.
- **src/models/** — Mongoose schemas (Content, Media, User, AuditLog). Keep index and static methods here.
- **src/repositories/** — optional data-access abstraction — helps testing and switching DB later.
- **src/middlewares/** — auth (JWT), RBAC, request validation, centralized error handler, rate limiter.
- **src/utils/** — logger, response formats, helpers (slugify, sanitize).
- **src/jobs/** — background tasks (scheduled publishes, search indexing, retry failed webhooks). They can run in same process or separate worker (prefer separate worker in prod).
- **src/integrations/** — wrappers for external services: Algolia, Cloudinary, SendGrid, Stripe, etc.
- **src/validators/** — request schemas (Joi/Yup) for endpoints.

- **src/tests/** — unit + integration tests. Use Jest + Supertest.
- **CHANGELOG.md** — maintain release notes.

Example important files (what to implement quickly)

- `src/config/index.js` — loads env and returns config object (e.g., `config.MONGO_URI`, `config.JWT_SECRET`, `config.FRONTEND_URL`, `config.WEBHOOK_SECRET`, `config.REVALIDATE_SECRET`).
- `src/config/db.js` — mongoose connection and graceful shutdown handlers.
- `src/api/v1/webhooks/webhook.controller.js` — checks `x-webhook-secret`, accepts `{ slug }`, calls frontend revalidate via `fetch(config.FRONTEND_URL + '/api/revalidate')`.
- `src/middlewares/auth.middleware.js` — extracts JWT from header/cookie and attaches `req.user`.
- `src/middlewares/error.handler.js` — catches thrown errors and sends formatted JSON error + logs.
- `src/jobs/publishScheduler.js` — checks `Content` with `publishAt <= now` and status=`scheduled`, publish and fire webhooks / indexers.
- `src/integrations/cloudinary.js` OR `src/integrations/s3.client.js` — handle upload, generate transformed URLs, store metadata.

Recommended npm packages

- core: `express`, `mongoose`, `dotenv`, `helmet`, `cors`, `body-parser`, `express-rate-limit`, `morgan` (or use winston/pino), `jsonwebtoken`.
- file upload: `multer`, `multer-storage-cloudinary` or `aws-sdk` + `multer-s3`.
- validation: `joi` or `yup`.
- logging: `winston` or `pino`.

- jobs: `bull` (Redis-backed) or `node-cron` for simple schedules.
- tests: `jest`, `supertest`.
- util: `slugify`, `node-fetch` (or built-in `fetch` in newer Node), `lodash`.

Env vars (examples to include in `.env.example`)

```
NODE_ENV=development
PORT=4000
MONGO_URI=mongodb://localhost:27017/headly
JWT_SECRET=your_jwt_secret
WEBHOOK_SECRET=webhook_secret
FRONTEND_URL=https://your-frontend.vercel.app
REVALIDATE_SECRET=revalidate_secret
CLOUDINARY_URL=cloudinary://key:secret@cloudname    # or S3 keys
ALGOLIA_APP_ID=
ALGOLIA_ADMIN_KEY=
SENDGRID_API_KEY=
REDIS_URL=redis://localhost:6379
```

Scripts (package.json) — suggested

```
"scripts": {
  "dev": "nodemon src/server.js --watch src",
  "start": "node src/server.js",
  "lint": "eslint .",
  "test": "jest --runInBand",
  "seed:admin": "node scripts/seed-admin.js",
  "migrate:media": "node scripts/migrate-media.js",
  "docker:build": "docker build -t headly-backend ."
}
```

Production considerations & separation of concerns

- In production split the **API** worker and **background worker** (jobs/indexer) into separate processes/containers. Jobs often need Redis (for Bull) and higher resilience.
- Keep webhook retries: when calling frontend revalidate, implement retry/backoff & record failures in `WebhookLog`.
- Run health checks and readiness endpoints for orchestration (Kubernetes / Render / Heroku).
- Use centralized logging + APM (Sentry, Datadog).

Testing & CI

- Unit test services with mocked repos.
- Integration test routes using an in-memory MongoDB (`mongodb-memory-server`) or a test DB.
- Add GitHub Actions: run lint → tests → build → optional deploy.

If you want, I can:

- generate the actual file skeletons (empty files + basic boilerplate) and produce a zip you can download, **or**
- create and populate the most important files now (`src/server.js`, `src/app.js`, `src/config/db.js`, `src/models/Content.js`, `src/api/v1/contents/contents.routes.js`), ready-to-run locally.