PROJECT GLOBAL BENE

Complete, production-ready playbook to build a **Global Bene** using **MERN (MongoDB, Express, React, Node)** + **AI/ML** (spam detection, content recommendations, moderation assistance) + **Data Analytics** (event pipeline, dashboards, experiments). Here is the architecture, modules, data models, ML designs, analytics pipeline, infra/CI, sprints, tasks, sample code snippets, KPIs, deliverables, and rollout plan. Use this as a blueprint you can hand to engineers, ML, data, and DevOps teams.

Project summary (one-liner)

Build **Global Bene** — a community discussion platform with communities, posts, threaded comments, upvote/downvote, moderation tools, AI-powered spam/misinfo detection and ranked recommendations, plus an analytics pipeline for product + moderation insights.

---

1. High-level architecture

Client (React) ⇄ API Gateway (Node/Express) ⇄ Services:

- Auth & User Service (JWT)
- Post/Comment Service
- Vote Service
- Community Service
- Moderation Service (human + AI)
- ML Service (spam classifier, recommendation engine)
- Analytics/Event Collector → Stream (Kafka/Redis streams) → Data Lake (S3) → Data Warehouse (Snowflake/BigQuery/Redshift) → BI (Metabase / Superset / Looker)

Storage & infra:

- Primary DB: MongoDB Atlas (documents for posts/comments); optional Postgres for relational joins
- Cache: Redis (rate-limit, hot-posts)
- Search: Elasticsearch (text search, ranking)
- Media: S3 / Cloudinary
- Realtime: Socket.IO (or WebSocket server)
- Queue / Background: BullMQ (Redis)

- Container: Docker → Kubernetes (EKS/GKE) or Render/Heroku for smaller teams
- CI/CD: GitHub Actions / GitLab CI
- Monitoring: Prometheus + Grafana, Sentry for errors, ELK for logs

---

2. Core modules & responsibilities (module-wise playbook)

Module A — Project setup & infra bootstrap

Goal: reproducible dev environment, CI, repo templates
Tasks

- Create monorepo or two repos (frontend, backend). Use `nx` or yarn workspaces if monorepo.
- Setup Dockerfiles, `docker-compose` for local dev (mongo, redis, backend, frontend).
- Create basic CI pipeline (lint, unit tests, build).
- Create environment variables template and secrets management plan.
  Deliverables
- `docker-compose.yml`, repo templates, CI pipeline.

Module B — Authentication & Authorization

Goal: secure user accounts, roles (user, mod, admin)
Backend tasks

- Implement JWT + refresh tokens, bcrypt password hashing.
- Email verification & password reset flows.
- Role-based middleware and route guards.
  Frontend tasks
- Login, signup, password reset UI, ProtectedRoute.
  APIs
- `POST /auth/register`, `POST /auth/login`, `POST /auth/refresh`, `POST /auth/forgot`, `GET /users/me`.
  Security
- Rate-limit auth endpoints; lockout on brute-force.
  Deliverables
- End-to-end auth flows, unit tests, docs.

Module C — Community (Subreddit) management

Goal: create/join/manage communities
Backend tasks

- Models: Community with name, title, rules, privacy, moderators.
- Endpoints: create, update, join, leave, moderator actions.
  Frontend
- Community creation flow, community page, join/leave button
  Deliverables

- Community CRUD + membership model, moderation UI.

Module D — Posts (core)

Goal: supports text/link/media posts, editing, deletion, ranking
Backend

- Post model, endpoints for CRUD, query filters (community, sort).
- Media upload (S3 signed URLs) and validation.
- Indexing: create Mongo indexes for fast feed queries.
  Frontend
- Composer UI (markdown support), previews, image upload.
  Deliverables
- Post creation flow with validations & tests.

Module E — Comments (threaded)

Goal: support nested threaded comments with pagination
Backend

- Comments model with parentId, depth, path (for efficient queries), numReplies counters, moderation flags.
- Endpoints for nested fetch (cursor-based).
  Frontend
- Comment composer inline, collapse/expand, optimistically render replies.
  Deliverables
- Threaded commenting system.

Module F — Voting & Ranking

Goal: robust upvote/downvote with scalability
Backend

- Separate `Votes` collection: `{ userId, targetType (post/comment), targetId, value }` with unique index on `(userId, targetType, targetId)`.
- Atomic score update: `db.collection.updateOne({_id}, {$inc: {score: delta}})`.
- Implement hot/top/new ranking algorithm (hot = score / time decay).
- Cache top N per community in Redis.
  Frontend
- Optimistic UI for votes; rate-limit interactions.
  Deliverables
- Vote accuracy & consistency; prevention of duplicate votes.

Module G — Moderation & Reporting

Goal: tools to detect & handle abuse, reports queue
Backend

- Reports collection: `reporterId, itemType, itemId, reason, status`.
- Moderator endpoints: review queue, remove content, ban user.
- Audit logs for all mod actions.
  AI-assisted features
- Auto-flag high spam/misinfo via ML model; priority-queue for moderators.
  Frontend
- Moderator dashboard, ban/soft-delete actions, reason templates.
  Deliverables
- Report queue + mod dashboard + audit logging.

Module H — ML Service (Spam Detection & Recommendations)

Goal: reduce spam, rank personalized feed, aid moderation
Subtasks

1. **Spam / Toxicity classifier**
   - Data: synthesize training dataset (public datasets: Jigsaw toxic comments, spam corpora) + platform-specific labeled data.
   - Model: fine-tuned transformer (DistilBERT / BERT) for text classification OR light model (Logistic Regression / XGBoost) for quick MVP.
   - API: `/ml/spam-predict` returning probability + explainability (top tokens).
   - Integration: run on new posts/comments server-side and tag or auto-queue for moderation.
2. **Recommendation engine**
   - Short-term: collaborative filtering + heuristics (community subscriptions, user history, post recency).
   - Medium-term: train a ranking model (DNN with features: user embeddings, post embeddings (SBERT), time decay, community).
   - Offline pipelines: batch feature generation and model training.
   - Online: serve recommendations from a model server (TF Serving / FastAPI + TorchServe) or use similarity search (FAISS) for content embeddings.
3. **Auto-moderation suggestions**
   - Suggest moderators which items to review (high spam score + high impact).
4. **Auto-tagging / flair suggestions**
   - Classify topic category for new posts (topic model / classifier).
     Deliverables

- ML service endpoints, training pipeline, monitoring for model drift, periodic retraining plan.

Module I — Analytics & Data Engineering

Goal: track product metrics, moderation metrics, ML metrics
Event model

- Standardize events (schema): `event_type`, `user_id`, `session_id`, `entity_id`, `entity_type`, `timestamp`, `props`.

  Pipeline
- Client → Event collector (REST or SDK) → Stream (Kafka/Redpanda or Kinesis) → Stream processing (Flink / Spark Streaming / Airflow for batch) → Data Lake (S3) → Data Warehouse (BigQuery / Snowflake) → BI (Metabase / Superset)

  What to track
- Product: daily active users (DAU), MAU, new signups, posts/day, comments/day, avg session length, retention.
- Engagement: votes/post, comments/post, time-to-first-comment.
- Moderation: flags/day, false positives, time-to-review, removal rates.
- ML: spam recall/precision, model score distribution.

  Analytics deliverables
- Base dashboards: Overview, Communities, Content Quality, Moderation Dashboard, Growth metrics.
- Experimentation setup: A/B testing using feature flags (LaunchDarkly / internal toggles).

  Data governance
- PII protection, GDPR compliance, retention policies, access controls.

Module J — Search & Discovery

Goal: fast full-text search & filters
Tasks

- Index posts/comments in Elasticsearch.
- Provide faceted search (community, author, date).
- Use search-result re-ranking by score / personalization.

  Deliverables
- Search API & UI with suggestions.

Module K — Realtime & Notifications

Goal: real-time updates for comments, votes, notifications
Tech

- Socket.IO or WebSocket layer, Redis pub/sub for scaling.

  Deliverables
- Live notifications panel, unread count, push notifications (optional).

Module L — Performance, Security & Observability

Tasks

- Rate limiting (Redis), CORS, helmet, input validation.
- Penetration testing and OWASP checklist.
- Instrumentation: OpenTelemetry, logs to ELK, errors to Sentry.

- Load testing (k6) to validate throughput.
  Deliverables
- Hardened security posture, observability dashboards.

---

3. Data models (concrete Mongo schemas — short)

User

```
{
  _id,
  username,
  email,
  passwordHash,
  roles: ['user','mod','admin'],
  joinedAt,
  bio,
  avatarUrl,
  karma: { posts:0, comments:0 },
  settings: {...}
}
```

Community

```
{
  _id, name, title, description, creatorId,
  rules: [], moderators: [], membersCount, isPrivate, createdAt
}
```

Post

```
{
  _id, communityId, authorId, title, body, media:[], url, tags:[],
  score:0, numComments:0, createdAt, updatedAt, status:
'active|removed|flagged', spamScore: 0.12
}
```

Comment

```
{
  _id, postId, authorId, parentId, body, score, path: 'rootId/childId/..',
createdAt, status
}
```

Vote

```
{ _id, userId, targetType: 'post|comment', targetId, value: 1|-1, createdAt
}
```

Report

```
{ _id, reporterId, targetType, targetId, reason, status, createdAt,
handledBy }
```

Event (analytics)

```
{ event_type, user_id, session_id, entity_type, entity_id, props: {...},
timestamp }
```

Indexes

- `posts`: `{ communityId:1, createdAt:-1 }`, `{ score:-1, createdAt:-1 }`, text index on `title, body`.
- `comments`: `{ postId:1, path:1 }`
- `votes`: unique index `(userId, targetType, targetId)`

---

4. ML design — spam detector & recommendation (detailed)

Spam / Toxicity Detector

Input: post.title + body + optional media metadata
Model options:

- MVP: TF-IDF + Logistic Regression or XGBoost trained on labeled data.
- Prod: fine-tune DistilBERT (fast) or BERT for classification.
  Labels: `spam`, `toxic`, `safe`, `nsfw`, `misinfo` (multi-label)
  Training pipeline:
- Data sources: public datasets (Jigsaw), in-house reported items, synthetically labeled items.
- Preprocessing: text clean, remove HTML, limit length, tokenizer.
- Features: embeddings (SBERT), metadata (account age, posts/day), behavior-based features.
  Serving:
- Containerized model server (FastAPI + TorchServe or TensorFlow Serving).
- Endpoint `/predict` returns `{ label_probs, explanation_tokens }`
  Integration:
- On post/comment creation: synchronous check. If spam probability > threshold1 => hold for review; > threshold2 => auto-remove or throttle.
  Monitoring:
- Track false positives, false negatives via moderation feedback.
- Retrain weekly/monthly as data accumulates.

Recommendation Engine

Phases:

1. **Heuristic (cold-start)**
   - Show posts from communities the user follows, plus trending in others.
2. **Content-based**
   - Use SBERT/Universal Sentence Encoder for post embeddings; recommend similar content.
3. **Collaborative Filtering**
   - Use implicit feedback (views, votes, comments) and matrix factorization for user embeddings.
4. **Learning-to-Rank**
   - Train ranking model (GBDT or neural) combining user, post, community, recency features.
     Serving

- Pre-compute top-K per user nightly; cache in Redis. For real-time, use FAISS for nearest neighbors on embeddings.
  A/B test
- Test variants (hot vs personalized) on engagement metrics.

5. Analytics pipeline (detailed)

Events tracked: page_view, post_create, comment_create, vote, community_join, report_create, mod_action, login, signup.
Collector: lightweight REST collector (node) writing to Kafka.
Streaming: Kafka → Stream processor (Spark/Flink) → real-time materialized views in Redis for dashboards.
Batch: ETL jobs in Airflow: aggregate daily/hourly metrics, feature generation for ML.
Warehouse: BigQuery / Snowflake / Redshift with partitioned tables by date.
Dashboards: Metabase / Superset for:

- Product Overview: DAU/MAU, new signups, sessions
- Engagement: posts/day, comments/day, votes/day, avg session length
- Moderation: flags/day, triage times, top reporters, false positive rate
- ML: model score distributions, drift alerts
  A/B test tracking: use event platform or internal experiment table; measure lift on core metrics.

---

6. CI/CD, infra and deployment

CI (GitHub Actions)

- On PR: run linters, unit tests, build.
- On merge to main: run integration tests, build images, push to registry.
  CD
- Deploy to staging via GitHub actions to k8s namespace.
- Manual approval for production; can gate with automated smoke tests.
  Kubernetes
- Use Helm charts for services (backend, ml, worker, socket).
- Setup horizontal pod autoscaler, resource requests/limits.
  Secrets
- Use Vault / AWS Secrets Manager.
  Backups
- DB daily backups with point-in-time restore.
  DR plan
- Multi-AZ deployment, infra as code (Terraform).

---

7. Security & compliance

- OWASP Top 10 mitigations.
- Sanitize markdown -> HTML (use DOMPurify).
- Rate limits; IP blocking for abuse.
- WAF in front; SSL everywhere; HSTS.
- Data retention and deletion flows (GDPR).
- Audit logs for mod/admin actions.

---

8. Testing strategy

- Unit tests (Jest for Node + React).
- Integration tests (Supertest, Cypress e2e).
- Contract tests for API (Pact).
- Load tests (k6): simulate 10k users on feed read, 1k concurrent post creates mock.
- Security testing: SAST, DAST.

---

9. Observability & SLOs

- Error budget & SLOs: e.g., 99.9% successful API responses per month.
- Instrumentation: traces for user-critical flows (post creation, voting).
- Alerts: error rate, CPU/memory spikes, queue length, model-server latency.

---

10. Team, roles & deliverables

Core team (small prod-ready)

- 1 Product Manager
- 2 Backend engineers
- 2 Frontend engineers
- 1 ML engineer / Data scientist
- 1 Data engineer
- 1 DevOps / SRE
- 1 QA engineer
- 1 Designer (UI/UX)
  Key deliverables per role
- PM: roadmap, acceptance criteria
- Backend: APIs, DB schema, tests
- Frontend: React app, components, tests
- ML: models, training pipeline, serving
- Data eng: ETL, warehouse, dashboards
- DevOps: infra, CI/CD, monitoring
- QA: test plans, automation

---

11. Sprint plan & timeline (recommended: 12-week roadmap)

(Adapt to team size; this is high-fidelity)

Phase A — Weeks 0–1 (Sprint 0: Setup)

- Tasks: repo, docker-compose, infra skeleton, basic auth flow
- Deliverables: dev environment, CI basics

Phase B — Weeks 2–5 (Core MVP sprints)

- Sprint 1 (Week 2): Auth, User profiles, basic UI skeleton

- Sprint 2 (Week 3): Communities + community pages
- Sprint 3 (Week 4): Post creation, feed (basic sort), media upload
- Sprint 4 (Week 5): Comments, threaded replies, vote logic

Phase C — Weeks 6–8 (Hardening & ML MVP)

- Sprint 5 (Week 6): Moderation flows & reporting, admin UI
- Sprint 6 (Week 7): Spam classifier MVP & integration
- Sprint 7 (Week 8): Recommendation MVP (heuristics + embeddings), search indexing

Phase D — Weeks 9–10 (Analytics & Stability)

- Sprint 8 (Week 9): Event pipeline, basic dashboards (growth + moderation)
- Sprint 9 (Week 10): Performance tuning, rate-limiting, load tests

Phase E — Weeks 11–12 (Polish & Release)

- Sprint 10 (Week 11): QA, security testing, documentation
- Sprint 11 (Week 12): Staging rollout, Beta with pilot users, production release

(If you need the 15-day compressed plan we discussed earlier, we can convert this; but for production-grade features 8–12 weeks is realistic.)

---

12. Acceptance criteria & QA checklist (example)

- Auth: users can register/login, email verification works, JWT expires & refresh works.
- Posts: create, edit, delete; images upload correctly; post visible in feed within 2s.
- Comments: threaded replies persist and render correct order.
- Votes: a user cannot vote twice; score updates atomically.
- ML: spam classifier precision > 0.8 on hold queue; recall prioritized to avoid misses.
- Analytics: DAU/MAU metrics computed daily and display in dashboard.

---

13. Sample code snippets (quickstart)

1. Post Mongoose schema (snippet)

```
const mongoose = require('mongoose');

const PostSchema = new mongoose.Schema({
  communityId: { type: mongoose.Types.ObjectId, ref: 'Community', required: 
true },
  authorId: { type: mongoose.Types.ObjectId, ref: 'User', required: true },
  title: { type: String, required: true, maxlength: 300 },
  body: { type: String },
  media: [{ url: String, type: String }],
  score: { type: Number, default: 0 },
  numComments: { type: Number, default: 0 },
  status: { type: String, enum: ['active','removed','flagged'], default:
'active' },
  spamScore: { type: Number, default: 0 }
}, { timestamps: true });

PostSchema.index({ communityId: 1, createdAt: -1 });
PostSchema.index({ title: 'text', body: 'text' });

module.exports = mongoose.model('Post', PostSchema);
```

2. Simple vote endpoint (concept)

```
router.post('/vote', auth, async (req, res) => {
  const { targetType, targetId, value } = req.body; // 1 or -1 or 0 to undo
  const userId = req.user.id;
  // upsert vote, compute delta
  const prev = await Vote.findOne({ userId, targetType, targetId });
  let delta = value;
  if (prev) {
    if (prev.value === value) { // undo
      delta = -value;
      await prev.remove();
    } else {
      delta = value - prev.value;
      prev.value = value; await prev.save();
    }
  } else {
    await Vote.create({ userId, targetType, targetId, value });
  }
  // atomic update on post/comment
  const Model = targetType === 'post' ? Post : Comment;
  await Model.updateOne({ _id: targetId }, { $inc: { score: delta } });
  res.json({ success: true });
});
```

3. Minimal spam check (sync)

```
// call model server
const resp = await axios.post('http://ml-service/predict', { text:
post.body });
if (resp.data.spam_prob > 0.9) {
  // soft block or hold
  post.status = 'flagged';
  post.spamScore = resp.data.spam_prob;
  await post.save();
  // push to report queue
  await Report.create({ reporterId: 'system', targetType:'post',
targetId:post._id, reason: 'auto-spam' });
}
```

14. KPIs & metrics to monitor

- Product: DAU, MAU, new signups/day, retention (D1/D7/D30).
- Engagement: posts/day, comments/post, votes/post, time per session.
- Moderation: flags/day, median time-to-action, false positive rate.
- ML: model latency, prediction throughput, precision/recall, concept drift.
- Infra: API latency (p95), error rate, CPU/RAM, queue backlog.

15. Deliverable checklist (MVP)

- Repo + docker-compose + CI
- Auth & profiles
- Communities
- Posts & media upload
- Comments (threaded)
- Voting
- Moderation UI & reports
- Spam classifier integrated
- Basic recommendation (personalized feed)
- Analytics pipeline & dashboards
- Search & indexing
- Staging + Production deployment with monitoring

TDL(To Do List)

Pick and produce it as per the project requirment :

- A runnable **starter repo** (backend + frontend skeleton) in JavaScript or TypeScript (I can generate files and zip).
- A **detailed 12-week sprint plan** broken into Jira-ready tickets with acceptance criteria.
- **ML model training notebook** (example DistilBERT fine-tune or TF-IDF+XGBoost) and API server code.
- **Analytics schema + Airflow DAGs** for ETL and an initial Metabase dashboard.
- **Helm charts + Terraform** skeleton for infra-as-code.

Module-wise Playbook — Detailed (actionable, team-ready)

Below is a **deep, implementation-ready playbook** for each module of the Reddit-clone (MERN + AI/ML + Analytics). Each module includes: purpose, scope, API surface, data model snippets, tasks (backend/frontend/ops), testing & QA checklist, security considerations, monitoring, acceptance criteria, owners, dependencies, and estimated effort (small-team, medium complexity). Use this to create Jira tickets or hand to engineers.

---

1. Authentication & Authorization

Purpose: secure user identity, session lifecycle, role-based access.
Scope / Features

- Register, email verification, login, refresh tokens, logout
- Password reset via email
- Roles: `user`, `moderator`, `admin`

- Account lockout, MFA (optional)
- Session revocation / device sessions list

APIs

- `POST /api/v1/auth/register` — { username,email,password }
- `POST /api/v1/auth/verify` — { token }
- `POST /api/v1/auth/login` — { usernameOrEmail,password } → accessToken, refreshToken
- `POST /api/v1/auth/refresh` — { refreshToken }
- `POST /api/v1/auth/logout` — { refreshToken }
- `POST /api/v1/auth/forgot` — { email }
- `POST /api/v1/auth/reset` — { token, newPassword }

Data model (snippet)

```
User {
  _id,
  username,
  email,
  passwordHash,
  roles: ['user'],
  emailVerified: Boolean,
  refreshTokens: [ { token, issuedAt, ip, device } ],
  failedLoginCount,
  lockedUntil,
  createdAt, updatedAt
}
```

Backend tasks

- Implement secure password hashing (bcrypt with cost factor).
- JWT access token (short TTL) + refresh tokens (stored hashed in DB).
- Email verification + secure token generation (HMAC or random UUID stored).
- Password reset tokens: single-use & expiry (e.g., 1 hour).
- Rate limiting & account lockout after N failed attempts.
- Middleware: `authRequired`, `roleRequired(['moderator'])`.

Frontend tasks

- Signup/login pages with validation and error handling.
- Persist access token in memory; refresh via silent refresh using refresh token in HttpOnly cookie or secure storage.
- Protected routes (redirect to login).
- UX flows: verify email screen, forgot/reset password.

Ops / Infra

- Setup email provider (SES, SendGrid) with templates.
- Secrets management for JWT secret & mail credentials.
- Monitoring: auth failure rate, suspicious IPs, lockouts.

Security

- Store refresh tokens hashed. Put refresh token in secure HttpOnly cookie or use rotation.
- Enforce strong password policy.
- CSRF protection for cookie flows.
- Rate limit endpoint: 10 req/min for auth endpoints, stricter for forgot.

Tests

- Unit tests for token handling.
- Integration tests for full auth flow (register → verify → login → refresh → logout).
- Security tests for SSRF/CSRF.

Acceptance criteria

- Users can register & verify email; login returns tokens; refresh works; logout invalidates refresh token.
- Brute-force protection in place.

Owner: Backend lead + Frontend lead
Effort: 4–6 days

---

2. User Profile & Account Management

Purpose: user preferences, avatars, karma, following.
Scope

- Profile view/edit, avatar upload, follow/unfollow users, display user posts & comments, user settings.

APIs

- `GET /api/v1/users/:username`
- `PUT /api/v1/users/:id` (auth)
- `POST /api/v1/users/:id/follow`
- `GET /api/v1/users/:id/posts`
- `GET /api/v1/users/:id/comments`

Data model
```
UserProfile {
  userId,
  displayName,
  bio,
  avatarUrl,
  socialLinks,
  karma: { post: 0, comment: 0 },
  followersCount,
  followingCount,
  createdAt
}
```

Backend tasks

- Profile endpoints & validation.
- Avatar upload pipeline: signed S3/Cloudinary uploads, thumbnail generation.
- Follow/unfollow logic with idempotence.
- Denormalize counts for fast reads (followersCount, postsCount).

Frontend tasks

- Profile page with tabs: posts, comments, about.
- Edit profile modal.
- Upload avatar with progress & crop UI.

Ops

- Thumbnails & CDN configuration.
- Storage lifecycle rules (optimize, retention).

Security

- Validate uploaded content type; virus scan (ClamAV) for enterprise.

Tests

- Unit tests, integration for follow/unfollow, avatar uploads.

Acceptance

- Profile updates persist; avatar displays; follow/unfollow atomic; counts consistent.

Owner: Frontend + Backend
Effort: 3–4 days

---

3. Community (Subreddit) Module

Purpose: group posts by topic, manage membership & moderators.
Scope

- Create, update, delete communities
- Join/leave; membership privacy (public/private)
- Moderation roles per community, community rules, flairs

APIs

- `POST /api/v1/communities`
- `GET /api/v1/communities`
- `GET /api/v1/communities/:name`
- `PUT /api/v1/communities/:id`
- `POST /api/v1/communities/:id/join`

- `POST /api/v1/communities/:id/leave`
- `POST /api/v1/communities/:id/mods` (assign mod)

Data model

```
Community {
  _id, name, title, description, creatorId,
  moderators: [userId], membersCount, members: [userId?],
  rules: [{id, text}],
  isPrivate: Boolean, createdAt
}
```

Backend tasks

- Create endpoints with validations (unique name).
- Member management + role enforcement.
- Invite/approval flow for private communities.
- Community flairs and rule enforcement.

Frontend tasks

- Community creation wizard.
- Member list & join/leave button.
- Community header & rule panel.

Ops

- Index community name (unique).
- Cache popular communities.

Tests

- Unique constraint tests; join/leave race condition tests.

Acceptance

- Communities can be created & joined; moderators can moderate members & posts; counts accurate.

Owner: Backend + Frontend
Effort: 3–5 days

---

4. Posts Module (Core)

Purpose: create and serve posts (text/link/media), support edit/delete, ranking.
Scope

- Post composer (Markdown or RichText), media uploads, edit/delete, pinning, locking, flairs, tags.

APIs

- `POST /api/v1/posts`
- `GET /api/v1/posts` — feed with filters & pagination
- `GET /api/v1/posts/:id`
- `PUT /api/v1/posts/:id`
- `DELETE /api/v1/posts/:id`
- `POST /api/v1/posts/:id/pin`
- `POST /api/v1/posts/:id/flair`

Data model

```
Post {
  _id, communityId, authorId, title, body, media:[], url, tags:[],
  score: 0, numComments: 0, createdAt, updatedAt,
  status: 'active|flagged|removed', spamScore: 0.12, flair: null
}
```

Backend tasks

- Create, edit, delete endpoints (auth & permissions).
- Media uploads: signed URL + background validation.
- Implement text sanitization (DOMPurify on server or sanitize html).
- Store denormalized fields (score, numComments).
- Implement feed query: filters (community, sort=hot/new/top), cursor pagination.
- Hot ranking algorithm: implement Reddit-like time decay.

Frontend tasks

- Post composer with image upload, link preview, markdown preview.
- Feed UI with infinite scroll or cursor-based pagination.
- Post details page (render markdown).

Ops

- Database indexes: `{ communityId:1, createdAt:-1 }`, `{ score:-1, createdAt:-1 }`, text index.
- Redis caching for top N posts per community.

Security

- Validate media sizes, types; virus scan.
- XSS protection for rendered HTML.

Tests

- Unit tests for create/edit/delete.
- Integration tests for feed ordering & pagination.

Acceptance

- Posts created appear in feed within SLA (e.g., <2 sec), media uploads succeed, edit/delete permissions enforced.

Owner: Backend + Frontend
Effort: 6–8 days

---

5. Commenting Module (Threaded)

Purpose: nested discussion, editing, deleting, vote per comment.
Scope

- Create comments (parentId optional), threaded view, edit/delete, collapse threads, lazy-load replies.

APIs

- `POST /api/v1/comments` — { postId, parentId, body }
- `GET /api/v1/posts/:postId/comments` — threaded (cursor/page)
- `PUT /api/v1/comments/:id`
- `DELETE /api/v1/comments/:id`

Data model

```
Comment {
  _id, postId, authorId, parentId, path: 'root/child', body,
  score:0, createdAt, updatedAt, depth
}
```

Backend tasks

- Implement path/depth approach: store `path = parent.path + '/' + id` for efficient subtree queries.
- Update `numComments` on Post atomically.
- Moderation flags per comment.

Frontend tasks

- Recursive CommentTree component with virtualization for long threads.
- Reply box inline, optimistic update.
- Collapse/expand UI.

Ops

- Index `postId` and `path` for fast retrieval.

Security

- Sanitize comment content, rate-limit comment creation.

Tests

- Test nested replies, deep thread ordering, performance for large numbers.

Acceptance

- Replies render in correct order, depth limit enforced (e.g., 10), performance acceptable for thousands of comments.

Owner: Backend + Frontend
Effort: 4–6 days

---

6. Voting & Reputation Module

Purpose: upvote/downvote mechanism and karma calculations.
Scope

- Per-post and per-comment voting, prevent double votes, undo votes, update score & user karma.

APIs

- `POST /api/v1/votes` — { targetType, targetId, value }
- `GET /api/v1/votes/:userId` — optional for history

Data model
```
Vote {
  _id, userId, targetType ('post'|'comment'), targetId, value (1|-1),
createdAt
}
```
Backend tasks

- Use separate `Vote` collection with unique index `(userId, targetType, targetId)`.
- Implement atomic delta updates: when vote changes, calculate `delta` and `$inc` on post/comment and update user's karma.
- Prevent race conditions: transactions (Mongo session) for multi-doc updates.
- Rate-limit voting actions to prevent abuse.

Frontend tasks

- Optimistic UI change; rollback on error.
- Visual state for current user's vote.

Ops

- Periodic job to reconcile votes vs stored score (data integrity check).

Security

- Detect vote fraud patterns (multiple accounts, same IP bursts) and flag for review.

Tests

- Unit & integration tests for vote transitions (no-vote → upvote → undo → downvote).

Acceptance

- Vote uniqueness enforced; post score reflects vote changes; no double counting.

Owner: Backend + Frontend
Effort: 3–4 days

---

7. Moderation & Reporting Module

Purpose: content quality & platform safety.
Scope

- User reporting, moderator queue, moderation actions (remove, warn, ban), audit logs.

APIs

- `POST /api/v1/reports` — { reporterId, targetType, targetId, reason }
- `GET /api/v1/mod/queue` — list flagged items
- `POST /api/v1/mod/action` — { actionType, targetId, reason, moderatorId }

Data model
```
Report {
  _id, reporterId, targetType, targetId, reason, status:
'open|reviewed|closed', createdAt, handledBy
}
AuditLog {
  _id, actionType, actorId, targetType, targetId, reason, metadata,
createdAt
}
```
Backend tasks

- Create report handling and mod action endpoints.
- Audit log for all mod/admin actions (immutable).
- Integrate auto-flagging via ML spam service.
- Notification to moderator (email/queue).

Frontend tasks

- Moderator dashboard with filters, bulk actions, searchable queue.
- Reporting UX in post/comment menu.

Ops

- RBAC for mod routes; log retention policy.

Security

- Rate-limit reports to prevent spam reports; protect mod endpoints.

Tests

- Tests for report lifecycle and mod actions; ensure audit logs created.

Acceptance

- Reports appear in moderator queue; actions recorded in audit log; moderators can perform expected actions.

Owner: Backend + Frontend + SRE
Effort: 4–6 days

---

8. ML Service — Spam Detection & Recommendations

Purpose: reduce spam/toxicity and personalize feed.

A. Spam/Toxicity Classifier

Scope

- Binary/multi-label classifier: spam/toxic/nsfw/misinfo
- Batch and online inference

APIs

- `POST /ml/predict/spam` — { text, metadata } → { label_probs, explain }

Design

- MVP: TF-IDF + XGBoost or lightweight transformer (DistilBERT) for higher accuracy.
- Input features: text, account age, posts/day, past flags.
- Thresholding: `>0.9 auto-remove`, `0.6-0.9 hold for review`.

Pipeline

- Data collection: initial seed from public datasets (Jigsaw) + platform labels.
- Training: reproducible scripts (notebooks & CI), model artifact versioning.
- Serving: FastAPI + Gunicorn or TorchServe, containerized.

Integration

- Synchronous call on post/comment creation (fast path) or asynchronous (background) depending on latency.

Monitoring

- Prediction latency, drift (distribution changes), false positive/negative rates.
- Feedback loop: moderation decisions feed back into training dataset.

Tasks

- Build training pipeline, model server, CI to package model, endpoints, logging.

Owner: ML Engineer
Effort: MVP 2–3 weeks; production hardening 4–6 weeks

B. Recommendation Engine

Scope

- Cold-start heuristics → embeddings-based ranking → learning-to-rank model

APIs

- `GET /ml/recommendations?userId=...` → list of postIds with scores

Design

- Phase 1: heuristics (community subscriptions + trending)
- Phase 2: content embeddings (SBERT) + FAISS nearest neighbours
- Phase 3: train ranking model with user-post features

Pipeline

- Batch feature generation, nightly recompute of top-K, cache in Redis.

Monitoring

- CTR on recommendations, downstream engagement metrics.

Owner: ML + Data Engineer
Effort: Phase 1 (1–2 weeks), Phase 2 (3–4 weeks), Phase 3 (4–8 weeks)

---

9. Analytics & Data Engineering

Purpose: event tracking, dashboards, model features, experiments.
Scope

- Event schema & collector, streaming to Kafka/Redis, ETL & warehouse, dashboards.

Event model (example)
```
{ event_type, user_id, session_id, entity_type, entity_id, props: {},
timestamp }
```
Pipeline Tasks

- Implement client-side event SDK (lightweight) + server collector.
- Stream to Kafka/Redpanda (or directly to S3 for small setups).
- Build ETL (Airflow) to populate data warehouse (BigQuery/Snowflake/Postgres).
- Create dashboards in Metabase/Superset: Product Overview, Engagement, Moderation, ML metrics.

Deliverables

- Event catalog, ETL DAGs, sample dashboards, baseline KPIs (DAU/MAU, retention, posts/day).

Security

- PII stripping & hashing before data lake ingestion.
- Access control for warehouse.

Owner: Data Engineer + PM
Effort: 3–5 weeks to production-grade pipeline

---

10. Search & Discovery

Purpose: text search, filters, suggestions.
Scope

- ElasticSearch indexing for posts & comments, autosuggest.

APIs

- `GET /search?q=...&filters...` (backend to query ES)

Tasks

- Index creation & mappings (title, body, tags, community).
- Implement suggesters & result highlighting.
- Re-rank by score + personalization.

Ops

- Monitor index health & reindex cadence.

Tests

- Search relevance tests, edge-case queries.

Owner: Backend + Data Engineer
Effort: 2–3 weeks

---

11. Realtime & Notifications

Purpose: live updates for comments, votes, notifications, unread counts.
Scope

- Socket.IO layer with Redis adapter for scaling; push notifications for mobile.

APIs

- WebSocket endpoints for subscriptions: `subscribe: post:{id}`, `notify:user:{id}`

Tasks

- Implement socket auth & room subscriptions.
- Emit events on create comment / vote / mod actions.
- Notification store & read/unread API: `GET /api/v1/notifications`, `PUT /api/v1/notifications/:id/read`.

Ops

- Redis pub/sub for multiple socket servers.
- Scaling plan & rate limiting.

Tests

- Load test for concurrent sockets, message delivery SLA.

Owner: Backend + SRE
Effort: 2–3 weeks

---

12. Media & File Uploads

Purpose: store images and other media reliably.
Scope

- Signed S3 uploads, thumbnails, CDN, content moderation pipeline (NSFW), retention.

APIs

- `POST /api/v1/uploads/sign` → return signed URL

Tasks

- Implement signed uploads & backend validate callback.
- Image processing (thumbnailing) via Lambda/worker (Sharp).
- CDN (CloudFront or Cloudinary) set up.

Security

- Limit allowed types, size, content checks.

Owner: Backend + DevOps
Effort: 2–3 days (basic); 1–2 weeks (advanced processing & scanning)

---

13. Admin Dashboard & Tools

Purpose: admin controls, analytics, user & content management.
Scope

- User management (ban/unban), content search & purge, metrics, mod assignment.

APIs

- Admin endpoints with RBAC; admin UI in a separate route.

Tasks

- Build admin UI with charts, search, bulk actions.
- RBAC enforcement and audit logs.

Owner: Backend + Frontend
Effort: 1–2 weeks

---

14. DevOps / CI-CD / Infra & Security

Purpose: reproducible infra, safe releases, observability.
Tasks

- Dockerize services; `docker-compose` dev setup.
- Kubernetes helm charts or managed PaaS manifests.
- CI pipeline: lint → test → build → push image.
- CD: staging auto-deploy; gated prod deploy with smoke tests.
- Setup monitoring: Prometheus/Grafana, ELK stack, Sentry.
- Backups & DR plan for MongoDB; automated backups to S3.
- Secrets management (Vault / AWS Secrets Manager).

Security Tasks

- WAF set up, TLS everywhere, CSP headers, HSTS.
- Regular dependency scanning & SAST.
- Pen-testing before production.

Owner: DevOps / SRE
Effort: Initial setup 2–4 weeks; ongoing work

---

15. QA, Testing & Release Management

Purpose: maintain quality and reliability.
Tasks

- Unit tests (Jest), integration tests (Supertest), e2e tests (Cypress).
- Contract tests (Pact), API schema validation (OpenAPI).
- Load testing (k6), security scans (OWASP ZAP).
- Release checklist: smoke test, canary deploy (k8s), rollback plan.

Owner: QA + DevOps
Effort: Integrated within sprints; dedicated 1–2 weeks for release prep

Cross-Module: Observability & SLOs

KPIs & Alerts

- API p95 latency < 500ms
- Error rate < 0.5%
- Spam classifier latency < 200ms
- DAU/MAU, retention D1/D7/D30
- Alerts: high error rate, queue backlog, model server downtime

Logs & Traces

- Structured logs (JSON) → ELK; distributed traces (OpenTelemetry).

Cross-Module: Data Integrity & Migration Strategy

- Use migration tool (migrate-mongo or custom scripts) with versioned migrations.
- Backout plan: reversible migrations where possible.
- Data reconciliation jobs (e.g., repair `numComments`, recalculating scores) scheduled nightly.

Developer Workflow & Sprint Example (12-week mapped to modules)

- **Sprint 0 (Week 0):** Repo, docker-compose, CI, basic auth.
- **Sprint 1 (Week 1):** Profiles + Communities.
- **Sprint 2 (Week 2):** Posts CRUD + media.
- **Sprint 3 (Week 3):** Comments + votes.
- **Sprint 4 (Week 4):** Moderation + reports + admin UI.
- **Sprint 5 (Week 5):** ML spam classifier MVP + integration.
- **Sprint 6 (Week 6):** Recommendations MVP + search indexing.
- **Sprint 7 (Week 7):** Notifications + realtime.
- **Sprint 8 (Week 8):** Analytics pipeline (collector & ETL).
- **Sprint 9 (Week 9):** QA hardening, load testing, security.
- **Sprint 10 (Week 10):** Beta rollout to pilot users.
- **Sprint 11 (Week 11):** Production launch readiness & docs.
- **Sprint 12 (Week 12):** GA release + wrap-up.

Sample Acceptance Criteria & Test Cases (generic)

- Auth: create user → verify email → login → access protected resource.
- Post: create post with image → appears in community feed within 2s → edit → delete.

- Comment: add reply → nested under parent → upvote increases comment score.
- Vote: same user cannot double vote; undo toggles value correctly.
- Moderation: report post → appears in mod queue → mod can remove & action logged.
- ML: flagged items appear with probability and mod actions feed into retrain dataset.

Deliverables per Module (minimum viable outputs)

- API documentation (OpenAPI), Postman collection
- Mongoose schemas / DB migration scripts
- Frontend components + unit tests
- Integration test suite
- Deployment manifests (Dockerfile + K8s/Helm or PaaS)
- Monitoring dashboards & runbooks
- Playbook for moderation & incident response

Final Recommendations & Next Steps

1. **Prioritize:** Start with Auth → Posts → Comments → Votes → Communities → Moderation → ML → Analytics.
2. **Start small:** build a working end-to-end MVP (basic feed + auth + comments) in 4–6 weeks with a 3–5 person team.
3. **Iterate ML:** integrate simple heuristics for spam first; add model once enough labeled data exists.
4. **Automate tests & CI** from day one; add observability early.

Perfect ✅ — Let's build a **Jira-ready, hierarchical ticket list (Epics → Stories → Tasks)** for the **Reddit Clone using MERN + AI-ML + Data Analytics**, aligned with Agile Scrum (6 Sprints × 2 weeks = ~12 weeks).
We'll structure it so you can **import directly into Jira, Trello, or ClickUp** using CSV or bulk issue creation.

🧩 Project: Reddit Clone – "GNCForum AI"

Stack: MERN (MongoDB, Express.js, React.js, Node.js) + Python (AI/ML microservices)
Duration: 12 weeks (6 sprints × 2 weeks)
Methodology: Agile Scrum
Teams:

- **Frontend Team (React)**
- **Backend/API Team (Node/Express)**
- **AI/ML Team (Python/Flask or TensorFlow)**
- **Data Analytics Team**
- **DevOps & QA Team**

🧱 EPICS, STORIES & TASKS (JIRA FORMAT)

EPIC 1: Project Setup & Architecture

1. **🎯 Goal: Build foundation of MERN + AI/Analytics architecture**

Type	Summary	Description	Owner	Sprint
Epic	Project Setup & Architecture	Setup environment, repos, project structure	Tech Lead	Sprint 1
Story	Setup GitHub Monorepo	Create monorepo with frontend, backend, ML folders	DevOps	Sprint 1
Task	Initialize React App	Setup with Tailwind + Vite + Redux Toolkit	Frontend	Sprint 1
Task	Setup Node.js API	Initialize Express app with routes folder structure	Backend	Sprint 1
Task	Configure MongoDB	Setup database connection, models folder	Backend	Sprint 1
Task	Setup Python ML microservice	Initialize Flask FastAPI microservice	AI/ML	Sprint 1
Task	Setup CI/CD pipeline	GitHub Actions for lint/test/build/deploy	DevOps	Sprint 1
Task	Setup Docker Containers	Dockerize all services	DevOps	Sprint 1

EPIC 2: User Authentication & Access Control

1. **🎯 Goal: Secure user login, signup, OAuth, and role-based access**

Type	Summary	Description	Owner	Sprint
Epic	User Auth	Implement full auth system	Backend	Sprint 2
Story	JWT-based Authentication	Signup, Login, Logout endpoints	Backend	Sprint 2
Task	Hash passwords	Use bcrypt + salt	Backend	Sprint 2
Task	Token refresh logic	Secure token refresh endpoint	Backend	Sprint 2
Story	Role-based Access	Admin, Moderator, User permissions	Backend	Sprint 2
Task	Middleware for authorization	JWT verification middleware	Backend	Sprint 2
Story	OAuth2 (Google/GitHub)	Social login	Frontend	Sprint 3
Task	Frontend auth pages	Signup/Login/Logout UI	Frontend	Sprint 2
Task	Profile Page	Display user profile, edit info	Frontend	Sprint 3

EPIC 3: Subreddit & Post Management

1. **🎯 Goal: Build the core Reddit-like functionality**

Type	Summary	Description	Owner	Sprint
Epic	Subreddit Management	Create and manage communities	Backend	Sprint 3
Story	Create Subreddit	Name, description, rules	Backend	Sprint 3
Task	Subreddit routes (CRUD)	Express routes for subreddit	Backend	Sprint 3
Task	UI for subreddit creation	React form + validation	Frontend	Sprint 3
Story	Post CRUD	Create, edit, delete, fetch posts	Backend	Sprint 4
Task	Upload images (Cloudinary)	Handle post media	Backend	Sprint 4
Task	Post display	Feed, single post view	Frontend	Sprint 4
Task	Comment system	Nested comments with replies	Backend	Sprint 5

EPIC 4: Voting, AI Moderation & Recommendation Engine

1. 🎯 **Goal: Implement post ranking, AI moderation & content recommendation**

Type	Summary	Description	Owner	Sprint
Epic	AI-Driven Features	AI moderation + recommendations	AI/ML	Sprint 5
Story	Upvote/Downvote System	Weighted scoring algorithm	Backend	Sprint 4
Story	AI Moderation	Use toxicity detection (BERT or Detoxify model)	AI/ML	Sprint 5
Task	Train moderation model	Dataset from Jigsaw or Reddit comments	AI/ML	Sprint 5
Task	Integrate moderation API	Flask → Node.js route bridge	AI/ML	Sprint 5
Story	Recommendation Engine	Personalized subreddit & post suggestions	AI/ML	Sprint 6
Task	Collect user activity logs	Feed training dataset	Data Analytics	Sprint 5
Task	ML model deployment	Deploy via FastAPI + Docker	DevOps	Sprint 6

EPIC 5: Data Analytics Dashboard

1. 🎯 **Goal: Build an analytics portal for user engagement, growth, trends**

Type	Summary	Description	Owner	Sprint
Epic	Analytics Dashboard	Visualize user & post data	Data Analytics	Sprint 6
Story	Collect interaction data	Upvotes, posts, comments, time spent	Backend	Sprint 6
Task	MongoDB Aggregation pipelines	Generate metrics	Backend	Sprint 6

Type	Summary	Description	Owner	Sprint
Story	Build dashboard UI	Charts with Recharts or Chart.js	Frontend	Sprint 6
Task	Data visualization endpoints	REST APIs for analytics	Backend	Sprint 6
Task	Predict engagement	ML model for user activity prediction	AI/ML	Sprint 6

EPIC 6: Testing, Security & Deployment

1. 🎯 **Goal: End-to-end QA, vulnerability checks, deployment to production**

Type	Summary	Description	Owner	Sprint
Epic	QA & Deployment	Ensure full quality & uptime	QA + DevOps	Sprint 6
Story	Unit & Integration Testing	Jest, Supertest	QA	Sprint 6
Task	Frontend Cypress tests	Simulate user flows	QA	Sprint 6
Task	API load testing	Postman + k6	QA	Sprint 6
Story	Security Audit	OWASP checklist + penetration testing	QA	Sprint 6
Story	Final Deployment	Deploy to AWS / Render / Vercel	DevOps	Sprint 6
Task	Setup monitoring	Prometheus + Grafana	DevOps	Sprint 6

🏁 MILESTONES

Milestone	Target Sprint	Description
MVP Ready	Sprint 4	Core Reddit functionality done
AI Features Complete	Sprint 5	Moderation + Recommendation integrated
Beta Release	Sprint 6	Analytics + Deployment complete
Production Launch	Post-Sprint 6	Live public version

📊 How to Import to Jira

1. Create a **CSV** with columns:
   `Issue Type, Summary, Description, Assignee, Sprint, Epic Link`
2. Paste the above table data per epic.
3. Import via Jira → **Project Settings → External System Import → CSV.**
4. Map "Epic Link" to connect stories to their parent Epic.

15-day fast-track plan (Initiation → Completion) for the **Reddit Clone Project (MERN + AI/ML + Data Analytics)** for **GNCIPL**, structured like a **mini Gantt chart in tabular format** for Agile rapid execution.
This plan assumes a **core development team of 6–8 members** (Frontend, Backend, AI/ML, Data Analytics, DevOps, QA) and focuses on delivering a **functional MVP** within 15 days.

🚀 GNCIPL Reddit Clone (AI-powered Community Platform) – 15-Day Development Plan

Day	Phase	Key Activities / Deliverables	Responsible Teams	Milestones / Output
Day 1	**Initiation & Setup**	Project kickoff, define objectives, assign roles, create GitHub repo & Slack channels	Project Manager, DevOps	✅ Project Charter Approved, Repos Created
Day 2	**Architecture & Environment Setup**	Setup MERN structure, initialize React + Node + MongoDB + Flask (AI service)	Backend, DevOps	✅ Base Architecture Ready
Day 3	**Authentication Module**	Build signup/login (JWT + bcrypt), setup auth middleware, basic UI for login/signup	Backend, Frontend	✅ Secure Auth System Working
Day 4	**User Profile & Access Control**	Create user profile page, role-based access (Admin/Moderator/User), profile CRUD	Frontend, Backend	✅ Profile Management Functional
Day 5	**Subreddit Module**	Create & Manage communities (CRUD), API routes for subreddit creation and joining	Backend	✅ Community System Ready
Day 6	**Post Management (Feed System)**	Create post APIs (Create/Edit/Delete), feed layout UI, Cloudinary integration for images	Frontend, Backend	✅ Feed System Functional
Day 7	**Comment & Vote System**	Nested comments API, UI integration, upvote/downvote logic	Backend, Frontend	✅ Post Interaction Features Ready
Day 8	**AI Moderation (Toxicity Detection)**	Integrate pretrained model (Detoxify/BERT), build Flask API for moderation	AI/ML Team	✅ Moderation API Live
Day 9	**Integrate AI Moderation in Posts**	Link moderation API to backend post routes, auto-flagging toxic comments	Backend, AI/ML	✅ AI Moderation Functional
Day 10	**Recommendation Engine (AI Personalization)**	Create simple collaborative filtering model for personalized feed	AI/ML, Data Analytics	✅ Personalized Feed Enabled

Day	Phase	Key Activities / Deliverables	Responsible Teams	Milestones / Output
Day 11	**Data Analytics Dashboard (Admin)**	Collect user activity logs, build dashboard using Chart.js/Recharts	Data Analytics, Frontend	✅ Analytics Dashboard Online
Day 12	**Testing & QA – Phase 1**	Unit testing (Jest/Supertest), UI bug fixing, API validation	QA, All Dev Teams	✅ QA Sign-off Phase 1
Day 13	**Deployment & Security**	Dockerize services, deploy to AWS/Vercel/Render, run vulnerability scan	DevOps, QA	✅ Live Beta Deployment
Day 14	**User Acceptance Testing (UAT)**	Internal demo, collect feedback, minor improvements, fix UI bugs	PM, QA, All Teams	✅ UAT Sign-off
Day 15	**Go-Live & Documentation**	Final release, monitoring setup (Grafana), product handover docs	DevOps, PM	🚀 Product Launched Successfully

---

🏁 Milestone Summary

Milestone	Day	Deliverable
M1 – Project Setup Complete	Day 2	All tech environments ready
M2 – Core Features Ready	Day 7	Auth + Subreddits + Posts + Comments
M3 – AI Integration	Day 10	Moderation + Recommendations active
M4 – Analytics & Testing	Day 12	Admin dashboard + QA validated
M5 – Beta Go-Live	Day 13	Deployed and accessible online
M6 – Final Launch	Day 15	Fully functional Reddit Clone released

---

⚙️ Team Roles & Responsibilities

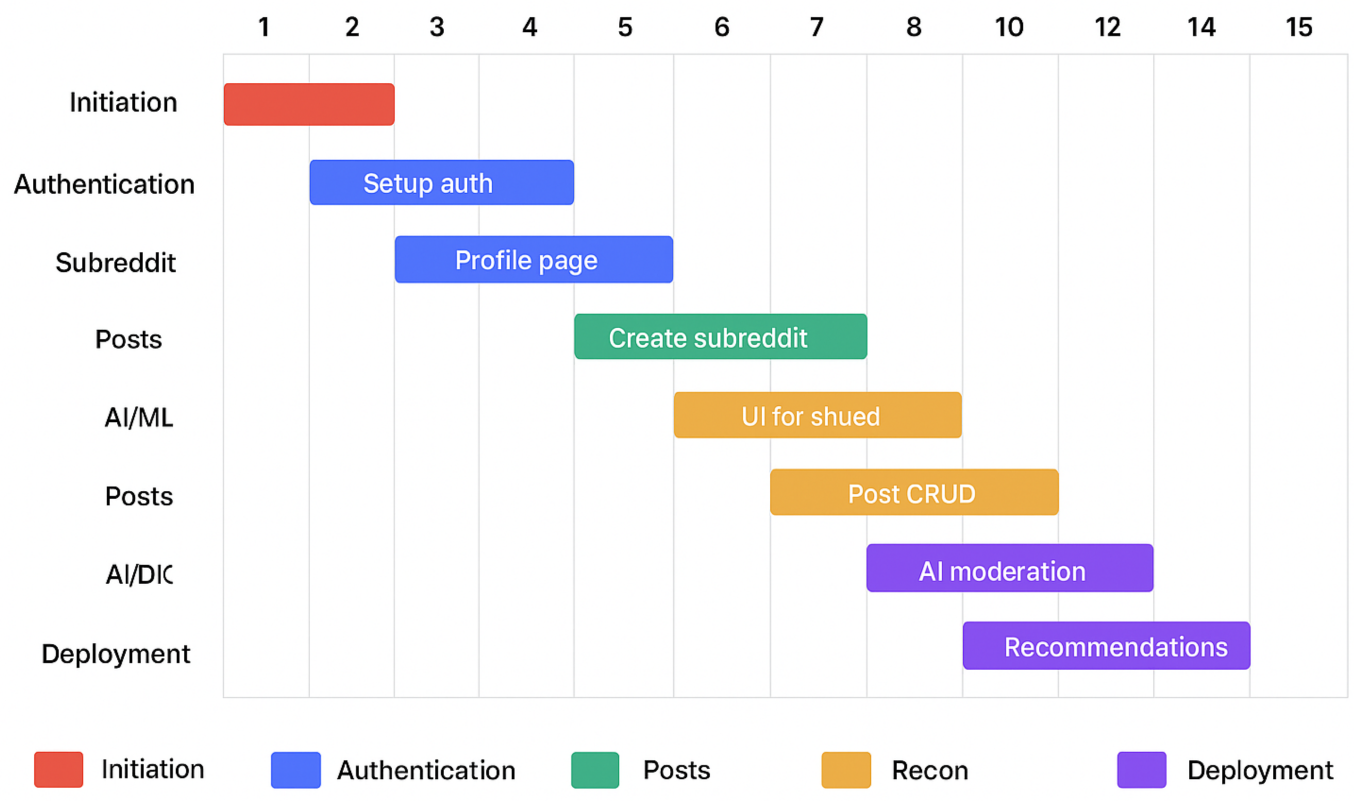
Role	Key Responsibilities
Project Manager	Track progress, manage Jira board, ensure deadlines
Frontend Developer	React UI, authentication, post feed, analytics dashboard
Backend Developer	Node.js APIs, MongoDB schema, authentication, moderation integration
AI/ML Engineer	Build & deploy AI moderation and recommendation services
Data Analyst	Collect logs, generate insights, visualize metrics
DevOps Engineer	Containerization, CI/CD, deployment, monitoring setup
QA Engineer	Testing (unit/integration/UAT), performance validation

---

📋 Jira Sprints for 15-Day Plan

Sprint	Duration	Goal	Deliverables
Sprint 1 (Day 1–5)	5 days	Foundation & Core modules	Auth, Profile, Subreddits
Sprint 2 (Day 6–10)	5 days	Content + AI	Feed, Comments, Moderation, Recommendations
Sprint 3 (Day 11–15)	5 days	Analytics + QA + Launch	Dashboard, Testing, Deployment

---



!!...This Part Is Left Blank Intentionally…!!