Research Grant Management — Software Development Document (SDD)

1. Document Version & Approvals

- **Version:** 1.0
- **Author:** Ibrahim Abdullahi (or project lead)
- **Date:** 2025-08-11
- **Approvals:** Project Sponsor | Product Owner | Technical Lead | Legal/Compliance

---

2. Purpose

This document describes the software requirements, architecture, design, and delivery plan for a **Research Grant Management Web Application** ("RGMA") to support the lifecycle of research grants from call and submission through review, award, funding disbursement, implementation (milestones), monitoring, reporting and closeout.

The SDD will be used by product owners, developers, QA, DevOps, and stakeholders to implement and validate the system.

---

3. Scope

In scope: - Grant calls & calls-for-proposals management - Application submission (project details, team, budgets, supporting documents) - Multi-stage review workflows (administrative check, peer review, panel review) - Budget and finance management (cost categories, approvals, disbursements) - Project milestone tracking and progress reporting - Metrics, dashboards and KPI reporting - Document repository and compliance artifacts (ethics approvals, MOUs) - Notifications & communications - Role-based access control, audit trail, and basic analytics

Out of scope (initial release): - Complex fund accounting (full ERP integration) - Advanced grant forecasting & investments - External sponsor portals (can be added later via API)

---

4. Stakeholders

- Product Owner / Grant Office
- Principal Investigators (PIs) / Researchers
- Co-Investigators / Research Teams
- Internal Reviewers / Peer Reviewers
- Finance / Grants Accounting

- Ethics & Compliance Office
- External Sponsors
- System Administrators / IT

---

5. User Roles & Permissions

- **Super Admin**: manages system config, user provisioning, roles
- **Grant Manager** (Admin at grant office): create calls, configure review workflows, assign reviewers
- **Researcher/PI**: create and submit proposals, upload documents, view award status, submit reports
- **Co-Investigator**: collaborate on proposal drafts
- **Reviewer**: access assigned proposals, submit reviews and scores, recommend funding
- **Finance Officer**: review budgets, approve disbursements, track payments
- **Ethics Officer**: review ethics documents, approve compliance
- **Sponsor Representative** (optional): view awarded projects and reports
- **Auditor**: read-only access to transaction logs and documents

Each role will map to a set of permissions. Use Role-Based Access Control (RBAC) with fine-grained resource-level controls.

---

6. Functional Requirements (High-level)

1. **Call & Program Management**
2. Create/close calls for proposals with deadlines, eligibility, templates and budget caps.
3. Publish and unpublish calls.
4. **Proposal Submission**
5. Multi-step web form (project summary, objectives, team, budget, milestones, attachments).
6. Save-draft and resume later.
7. Attachment upload (PDF, docx, images) and virus scanning.
8. **Review Workflow**
9. Administrative check (auto-check required fields) before peer review.
10. Blind or open peer-review options.
11. Scoring rubrics per call; reviewer assignments configurable.
12. Panel moderation and final decision recording.
13. **Budget & Finance**
14. Capture budget line items (personnel, equipment, travel, overhead, indirects).
15. Finance validation rules and approval routing.
16. Track disbursement schedules and payment records.
17. **Award Management**
18. Issue award letters, accept/decline flows.
19. Generate award agreements (templated documents with merge fields).
20. **Project Monitoring**
21. Milestone creation, progress submission, deliverable upload.
22. Activity logs and PI narratives.
23. **Reporting**
24. Standard reports (award summary, spend vs budget, overdue deliverables).

25. Dashboards for managers and PIs.
26. **Document Management**
27. Central repository with versioning and access controls.
28. **Notifications & Communication**
29. Email templates and in-app notifications for deadlines, review invites, decisions.
30. **Audit & Compliance**
31. Full audit logs for actions and file uploads.
32. **Search & Filters**
33. Full-text search and advanced filters (by PI, call, status, keywords).
34. **APIs & Integrations**
35. REST API for external systems (HR, finance, SSO, institutional CRIS systems).

---

7. Non-Functional Requirements

- **Security**: OAuth2 / OpenID Connect SSO; password policies; MFA; data encryption at rest and transit; role-based access control.
- **Privacy/Compliance**: support for data retention policies and ability to anonymize personal data on request. Follow GDPR-like principles if applicable.
- **Availability**: 99.5% SLA for the application during business hours.
- **Performance**: page load < 2s for dashboards; API response < 500ms under normal load.
- **Scalability**: horizontally scalable backend and stateless web servers; database scaling plan.
- **Backup & Recovery**: daily backups with 30-day retention, RPO/RTO specified.
- **Maintainability**: modular codebase, clear API contracts, test coverage >80% for critical modules.
- **Accessibility**: WCAG 2.1 AA compliance for main user flows.
- **Internationalization**: support English initially; design for future localization.

---

8. Data Model (Core Entities)

Below are core entities and key attributes. A normalized relational schema (PostgreSQL recommended) is a good fit.

- **User**: id, name, email, role_ids, department, phone, institution_id, last_login
- **Institution**: id, name, address, type
- **CallForProposals**: id, title, description, open_date, close_date, budget_cap, eligibility, rubrics
- **Proposal**: id, call_id, title, abstract, status, pi_user_id, submission_date, total_budget
- **ProposalTeamMember**: id, proposal_id, user_id, role, contribution
- **BudgetLine**: id, proposal_id, category, description, amount
- **Attachment**: id, proposal_id, filename, filetype, size, url, version
- **Review**: id, proposal_id, reviewer_id, scores(json), comments, date_submitted
- **Decision**: id, proposal_id, decision_type (award/reject), rationale, approved_by
- **Award**: id, proposal_id, award_amount, start_date, end_date, disbursement_schedule(json)
- **Disbursement**: id, award_id, amount, status, payment_date, transaction_ref
- **Milestone**: id, award_id, title, due_date, status, deliverable_attachment_id
- **Report**: id, award_id, reporting_period, narrative, attachments

- **AuditLog**: id, user_id, action, resource_type, resource_id, timestamp, metadata

---

9. Example API Endpoints (RESTful)

- `POST /api/v1/auth/login` — authenticate (or use OIDC)
- `GET /api/v1/calls` — list calls
- `POST /api/v1/calls` — create call (Grant Manager)
- `GET /api/v1/calls/{id}` — call details
- `POST /api/v1/proposals` — create proposal (save draft)
- `PUT /api/v1/proposals/{id}/submit` — submit proposal
- `GET /api/v1/proposals/{id}` — view proposal
- `POST /api/v1/proposals/{id}/attachments` — upload file
- `GET /api/v1/proposals/{id}/reviews` — get reviews
- `POST /api/v1/proposals/{id}/reviews` — submit review
- `POST /api/v1/awards` — create award record
- `GET /api/v1/reports/awards` — export award report

Use pagination, filtering query parameters, and consistent error codes (RFC7807 problem details recommended).

---

10. UI/UX — Pages & Flows

Main UI sections: - Landing / Public calls page - Login / Registration / SSO - Dashboard (role-specific) — KPIs, pending actions - Call creation wizard - Proposal builder (multi-step) - Review panel (reviewer interface) - Award management area - Finance dashboard - Document repository - Admin settings (users, roles, system configs)

Key UX considerations: - Save-draft often and warn before autosave conflicts - Provide progress indicators for multi-step forms - Make submission requirements explicit and show attachment checklist - Reviewer UI must be uncluttered with clear rubrics and blind-review toggles

---

11. Architecture & Components

Suggested architecture (3-tier, cloud-ready): - **Frontend:** React (TypeScript) + Tailwind CSS. Component library for consistency. - **Backend API:** Node.js (NestJS or Express) or Python (FastAPI) — RESTful JSON APIs. - **Database:** PostgreSQL for relational data. - **File Storage:** AWS S3 (or equivalent) for attachments, with signed URLs. - **Cache:** Redis for sessions, rate-limiting, and small caches. - **Authentication:** OpenID Connect / OAuth2 (Keycloak, Auth0, or institution SSO). - **Queue:** RabbitMQ or AWS SQS for background tasks (email, virus-scan, document generation). - **Search:** Elasticsearch or Postgres Full-Text for search. - **CI/CD:** GitHub Actions / GitLab CI to build, test and deploy containers. - **Containerization:** Docker; orchestration via Kubernetes (or ECS) for production. - **Monitoring:** Prometheus + Grafana; centralized logs (ELK Stack or hosted alternatives).

Deployment: Infrastructure as Code (Terraform) to provision networks, DB, buckets, and k8s cluster.

---

12. Security Design

• Use HTTPS everywhere (TLS 1.2+).
• JWT short-lived tokens + refresh tokens if using token flows.
• Role-based authorization checks on every endpoint and resource.
• Input validation + parameterized DB queries to prevent SQL injection.
• File scanning (antivirus/AV) on upload; limit file types and sizes.
• S3 bucket policies: private by default; signed URLs for downloads.
• Content Security Policy (CSP) to protect against XSS.
• Logging and SIEM integration for suspicious activity.
• Regular security testing: SAST, DAST, dependency vulnerability scans.

---

13. Data Migration & Integration

• Provide import tools for historical award/proposal data (CSV, JSON).
• Define mapping tables and data validation steps.
• Implement idempotent import scripts and staging area.
• Provide webhook/API for integrating with HR, finance, or institutional CRIS.

---

14. Testing Strategy

• **Unit tests**: all business logic (target 70–90% coverage depending on module criticality).
• **Integration tests**: database, API, file storage interactions.
• **End-to-end tests**: critical user journeys (proposal submission, review, award).
• **Performance tests**: load testing for API and DB.
• **Security tests**: automated dependency checks + periodic pen tests.
• **Accessibility tests**: automated audits (axe) + manual validation.

---

15. Deployment & Release Plan

Environments: dev -> staging -> production - Automated pipelines to deploy to dev on merge, staging on release branch, production on tag. - Feature flags for risky features to do gradual rollout.

Rollback plan: keep last two release images and db migrations reversible. Backups available.

---

16. Monitoring & Observability

- Application metrics (uptime, latency, error rates)
- Business metrics (number of submissions, outstanding reviews, award pipeline)
- Alerts on error rate spikes, slow responses, failed background jobs

---

17. Project Plan (High-level)

Assumption: small cross-functional team (Product Owner + 2 backend devs + 2 frontend devs + 1 QA + 1 DevOps) — 12 weeks MVP.

Phase 0 — Discovery (1 week) - Finalize requirements, wireframes and acceptance criteria.

Phase 1 — Core backend & DB + Authentication (2 weeks) - Setup infra, DB schema, user auth, basic user management.

Phase 2 — Proposal submission & attachments (2 weeks) - Proposal multi-step form, save-draft, upload and store files.

Phase 3 — Review flows & scoring (2 weeks) - Reviewer assignment, review UI, scoring, admin panel.

Phase 4 — Awards & Finance basics (2 weeks) - Award records, disbursement tracking, finance validation.

Phase 5 — Dashboards, reporting, testing (2 weeks) - Manager dashboards, key reports, e2e testing and bug fixes.

Phase 6 — Hardening & Launch (1 week) - Security review, performance tune, production deployment.

Deliverables per phase: working build, API docs (OpenAPI), user guide, test report.

---

18. Acceptance Criteria (Example)

- Users can register/login via SSO and perform role-allowed functions.
- A PI can submit a fully validated proposal with attachments and receive confirmation.
- Reviewers can access assigned proposals and submit scored reviews.
- Finance officer can view and approve budget items.
- Audit logs persist and can be queried by Admin.
- Backup and restore tested in staging.

---

19. Risks & Mitigations

- **Complex institutional integrations** — Mitigation: define minimal viable integration (SSO + CSV import) and schedule deeper integrations later.
- **Data sensitivity** — Mitigation: strict access controls, encryption, and retention policies.
- **Reviewer adoption** — Mitigation: easy reviewer UI, short training materials, offline review templates.
- **Budget miscalculation** — Mitigation: validation rules and finance approval gates.

---

20. Operational & Support Considerations

- SLA for bug fixes and support channels
- User onboarding materials, help center articles, short videos
- Admin tools for user management and audit exports

---

21. Appendices

A — Example ERD (textual)

- User (1) — (many) ProposalTeamMember
- CallForProposals (1) — (many) Proposal
- Proposal (1) — (many) BudgetLine
- Proposal (1) — (many) Attachment
- Proposal (1) — (many) Review
- Proposal (1) — (one) Decision
- Decision (1) — (one) Award
- Award (1) — (many) Milestone
- Award (1) — (many) Disbursement

B — Example Review Rubric (JSON)

```
{
  "criteria": [
    {"id":"impact","weight":40,"scale":10},
    {"id":"feasibility","weight":30,"scale":10},
    {"id":"team","weight":20,"scale":10},
    {"id":"budget","weight":10,"scale":10}
  ]
}
```

C — Example Database Indexing Notes

- Index `proposals(call_id, status)` for common queries

- GIN index on `to_tsvector` for full-text search on titles and abstracts

---

22. Next Steps / Recommendations

1. Approve SDD and prioritize features for the MVP (which features to include in 12-week plan).
2. Produce low-fidelity wireframes and a clickable prototype for stakeholder feedback.
3. Prepare OpenAPI (Swagger) skeleton and start backend scaffolding.
4. Select SSO provider and hosting/cloud provider.
5. Plan a 2-week sprint cadence with demos at the end of each sprint.

End of document — any diagrams (UML, ERD) can be produced on request. If you'd like this exported as a Word/PDF or converted to a JIRA-ready backlog with user stories and tasks, I can generate that next.