Grant Guru

Colby Wirth
James Tedder
Abdullahi Abdullahi
Mathieu Poulin

Project Overview & User Features

Project Goal

To create a "Grant Guru" system that allows users to efficiently find, track, and manage grant applications.

Core Features

Account Management & Grant Tracking

- Users can create personal accounts to serve as a centralized hub for all their grant-related activities.
- Once logged in, users can **search for grants**, create applications, and track their status.
- The system allows for storing application documents and personal notes.

Proactive Grant Alerts

- Users can opt-in to receive **email notifications** for newly posted or forecasted grants that match their interests.
- Notification frequency is customizable (e.g., daily, weekly, or monthly).

Project Overview & User Features

Project Goal

To create a "Grant Guru" system that allows users to efficiently find, track, and manage grant applications.

System Requirements

Performance

- **Fast Local Queries:** User actions, like loading application data, must complete in **under 2 seconds.**
- **Intelligent Caching:** The system prioritizes searching its own local database first. It only queries external sources if the data is missing, which minimizes wait times and external requests.

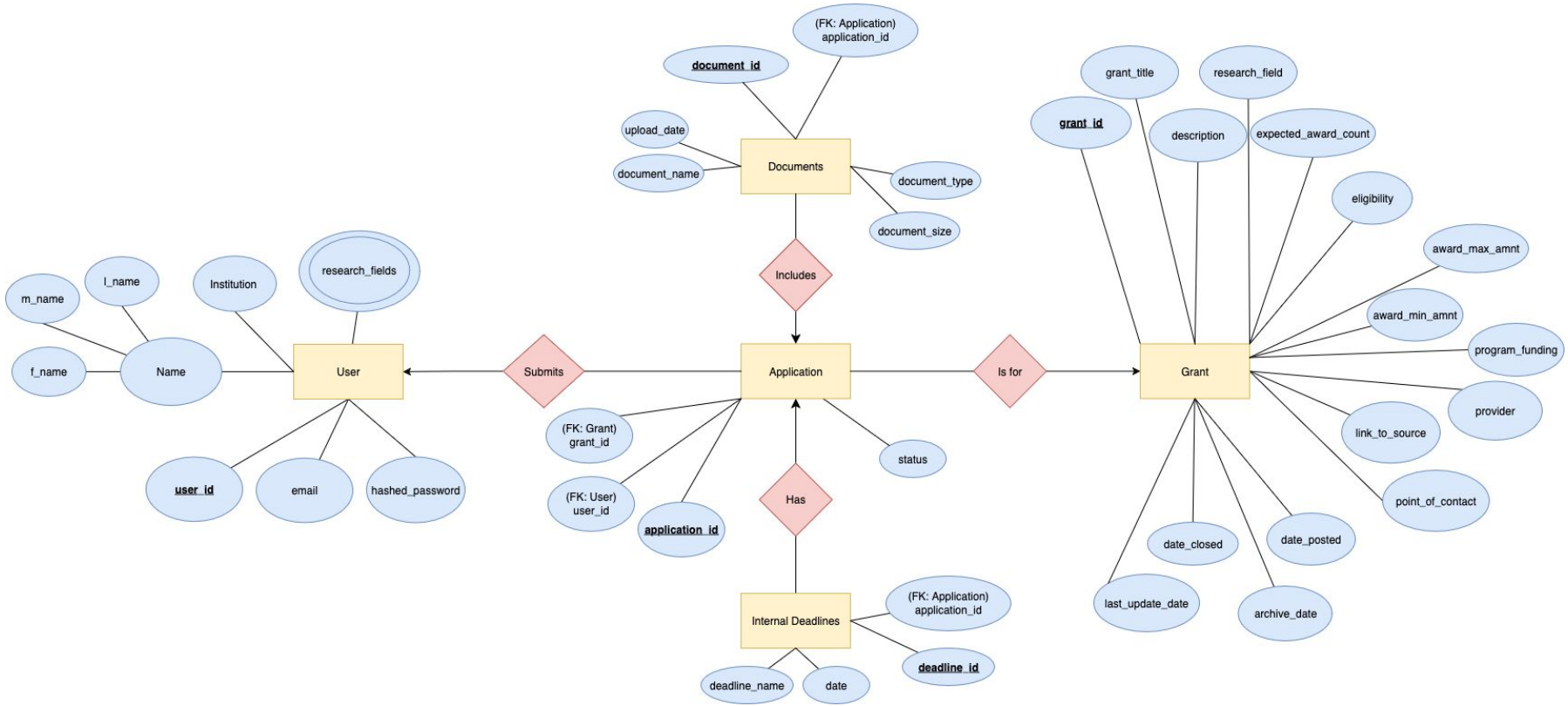
Usability

- The interface is designed to be intuitive for anyone with basic experience using web applications.

Security

- **Data Isolation:** Security is enforced at the database level. **Database views** and **role-based permissions** will be used to ensure a user can only ever access their own data.
- **Encryption:** All sensitive user information, especially passwords, must be stored in an **encrypted** format in the database
- **Rate Limiting:** To prevent abuse, external data scraping is limited to one request every five seconds per user.

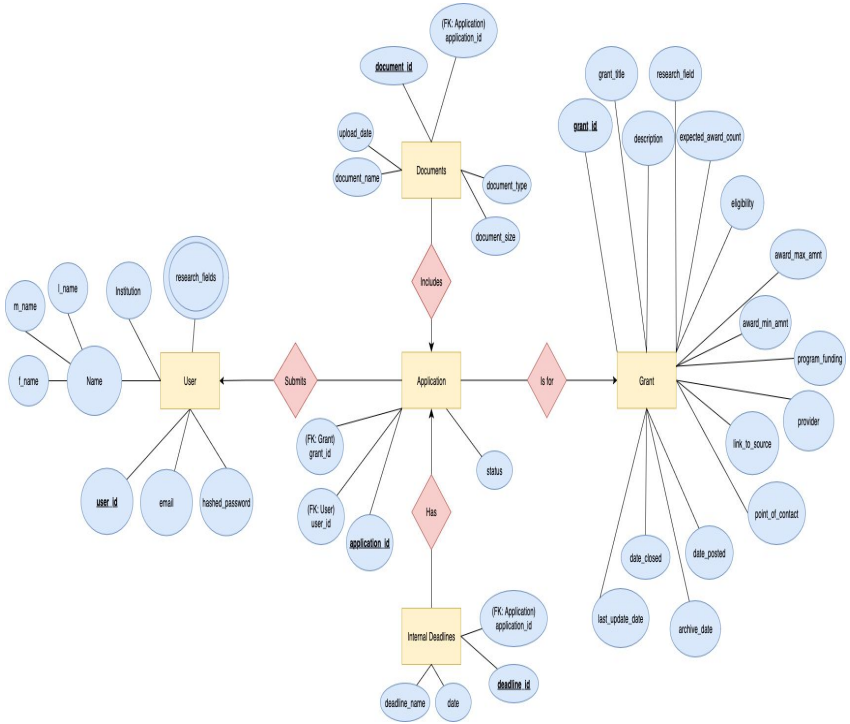
ER diagram


(FK: Application)
application_id
document_id
upload_date
document_name
Documents
document_type
document_size
Includes
grant_title
research_field
grant_id
description
expected_award_count
eligibility
award_max_amnt
award_min_amnt
program_funding
provider
link_to_source
point_of_contact
date_closed
date_posted
last_update_date
archive_date
Grant
Is for
m_name
l_name
Institution
research_fields
f_name
Name
User
Submits
Application
(FK: Grant)
grant_id
status
(FK: User)
user_id
application_id
Has
user_id
email
hashed_password
Internal Deadlines
(FK: Application)
application_id
deadline_id
deadline_name
date

Normalization Analysis

✅ How We Achieved BCNF

- **1NF (Atomicity):** All attributes are atomic (indivisible), and composite attributes like `Name` have been properly decomposed (`f_name`, `l_name`).
- **2NF (No Partial Dependencies):** We used single-attribute primary keys throughout the design, which inherently prevents the existence of partial dependencies.
- **3NF (No Transitive Dependencies):** All non-key attributes depend directly on their primary key, not on other non-key attributes. For example, `institution` depends directly on `user_id`, not transitively through another attribute like `email`.
- **BCNF (Determinants are Superkeys):** Every functional dependency has a superkey on the left-hand side, satisfying the requirements for BCNF.



Normalization Analysis

❌ Why We Didn't Achieve 4NF

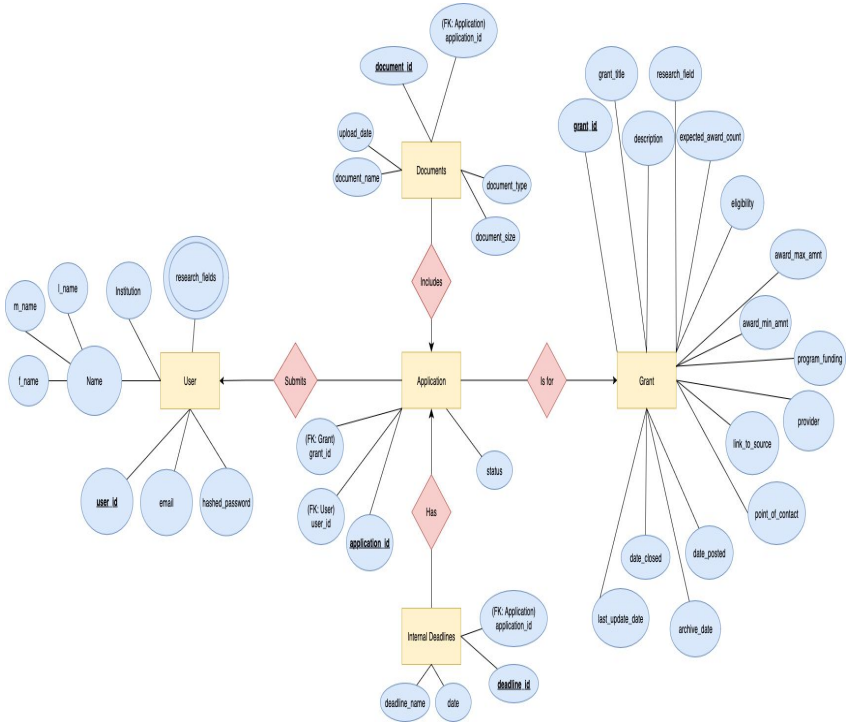
The design violates 4NF due to a **multi-valued dependency (MVD)** in the `User` entity:

- **The MVD:** `user_id` →→ `research_fields`
- **The Problem:** A user can have multiple independent research fields (e.g., 'Machine Learning', 'NLP'). Storing these in the same table would cause redundant repetition of user data (like `email` and `institution`) for each field, which violates 4NF.

🚀 Path to 4NF

To achieve 4NF, we would decompose the `User` entity into two separate tables to isolate the multi-valued attribute:

- `User(user_id, email, hashed_password, f_name, l_name, institution)`
- `UserResearchField(user_id, research_field)`




THX