**ARTISTREE**

**Project Direction Overview:**

I would like to create a database for a webapp which will help hire creative freelancers and also help freelancers choose projects to work on. There are many freelancing apps out there which is a hub for all sorts of professions, but this app targets only those who work in the creative field. Such professions include musicians, sound designers, writers, lyricists, costume designers, makeup artists, jewelry makers etc. An employer can post a project online and put out RFPs on the messageboard that will contain certain skill names in the form of tags and freelancers will be suggested those based on the skills they have selected for themselves when they created their profile on the website. Freelancers can even sign contracts with their employers and keep those records on the website. Employers can also pay their recruits through the portal on the website and the website will keep track of when the payment is due and how much needs to be paid on a periodic basis based on what was agreed upon in the contract.

For example, Employer Joseph Calamaza is organizing a corporate party for an account firm and he is looking for a stand up comedian for the entertainment of the group. He posts a request for comedians who are eager to perform in front of a corporate audience at said place. He also puts down tags like comedian, corporate gala, and <name of place>. Comedian and freelancer Sandra Bullington responds to his request. Joseph looks into her profile to see her experiences and he seems impressed. He contacts her and lets her know about the details of the event and talks price with her. Once Sandra agrees, he creates a contract on the app and fills out a form where he has to mention when Sandra needs to be paid and how much amount. Those information, once signed by employing authorities, will be sent over to Sandra for her to go through and sign and send back. Both parties will have the same copy that will not be able to be edited further. Based on that, Sandra will be paid her due amount on the day it is due through the app which will already have had Sandra’s payment information. Sandra will also be rated based on her performances and Joseph can even write her a recommendation. Sandra can also rate Joseph as an employer so that more people can be helped in deciding whether to work with him or not.

This app will be called Artistree because it is like a tree with many branches of freelancing activities.

**Use Cases and Fields:**

1. **Creating an account as an employer or a freelancer - Use Cases:**

* A user enters the website for the webapp.
* They are asked to log in or sign up.
* They enter their credentials: Name, Address, Phone Number, Email, Employer/Freelancer etc.
* Account is created for them.

Significant fields for this use case are given as follows:

| **Field** | **What it Stores** | **Why it’s Needed** |
| --- | --- | --- |
| Username | This is the name using which user will login to the system every time. | Users need a credential to create their digital signature on the website. This is one of the values for that purpose. |
| Password | This is the password to secure their data. | Users need a credential to create their digital signature on the website. This is one of the values for that purpose. |
| FirstName | This is the first name of the user | This is needed to display the user’s name in any form or transaction. |
| LastName | This is the last name of the user | This is needed to display the user’s name in any form or transaction. |
| PhoneNumber | This is the user’s contact details. | For easier reachability to the user and for OTP purposes. |
| Email | This is the user’s contact details. | For easier reachability to the user and for OTP purposes. |
| Address | This is the user’s contact details. | Sending bills and storing basic contact information. |
| AccountType | Employer/Freelancer | This will denote what the dashboard will look like for the user. |

1. **Creating skills - Use Cases:**

* A freelancer will get a link on their dashboard that says Skills
* They will get a form that will let them add a skill
* They will also be able to add a type to that skill. It will be like a genre. So if a person can play guitar, their skill will be Playing guitar and it will go under the type: Music. This will help garner more music playing related jobs for them.

Significant fields for this use case are given as follows:

| **Field** | **What it Stores** | **Why it’s Needed** |
| --- | --- | --- |
| SkillName | This will be the name of the skill | If the user wants to include a skill that is not already in the system, they will be able to add it. |
| Description | This is the description if the user wants to add any. | This will be used to further describe the skill. |
| Type | This is like a genre for skills like: music, literature, dancing etc. | This will be needed to provide relatable jobs to freelancers. |

1. **Creating Job/Project - Use Case (REVISED):**

* Employer creates a job or a project.
* They will post it with the name, venue, other supporting URLs etc for the job.
* The creator of that project will be able to assign a freelancer to it and the project will change its status so that it is no more publicly visible.
* Now the Project is available to both the employer and the assigned freelancer.
* Once the job is done, the employer can mark it as completed.

Significant fields for this use case are given as follows:

| **Field** | **What it Stores** | **Why it’s Needed** |
| --- | --- | --- |
| ProjectTitle | This will be the title of the project | This will be needed to display the entire project on the message board. |
| Description | This will be additional descriptions of what the project/job is like. | One the project view is opened, the description will give more details about the job/project. |
| Venue | This will be where the job/project will take place. | This will be needed if the job is to be held somewhere physical and not online. |
| Creator | This will be the name of the user who created this project. | Employer/Freelancer who created the job. |
| Tags | These will be tags that define the project | Needed to seek the attention of appropriate freelancers. |
| Status | What point of the timeline the project is now standing | This will be needed to track the progress of the job and to make sure who is having access to it. |

1. **Create Contract - Use Case:**

* Once there is an agreement between an employer and freelancer, the employer will create a contract.
* They will be able to add the project title to the contract, enter the name of the employer, freelancer, enter details of the contract and the payment details on it.
* The contract, once confirmed, will be sent to the freelancer to read and sign.
* They will send it back to the employer and a copy of both the signed contract will remain on both profiles.

Significant fields for this use case are given as follows:

| **Field** | **What it Stores** | **Why it’s Needed** |
| --- | --- | --- |
| EmployerName | Name of employer | This is to document who the employer is. |
| FreelancerName | Name of freelancer | This is to document who the freelancer is. |
| ProjectTitle | Title of project | This is to document which project the contract is for. |
| ContractBody | Description of the contract and other important details | Documentation of the details of the contract. |
| CreateDate | Date of creation of the contract | This is needed to know when the contract was created. |
| SignedDate | Date of signing of the contract | This is needed to know when both parties signed the contract. |
| PayableAmount | Amount to be paid to the freelancer | This will be needed to document and track the payment amount. |
| PaymentDate | Due date of payment | This will be needed to remind the payer when to pay the payee. |

1. **Pay freelancer - Use Case:**

* On the due date of the payment, the employer will send the money to the freelancer.
* This amount will be tracked using the payment value input in the contract.
* The employer will use the billing details that will be associated with the freelancer’s profile and send the money over to them.
* The receipt will remain in the system and users can opt for email options as well

Significant fields for this use case are given as follows:

| **Field** | **What it Stores** | **Why it’s Needed** |
| --- | --- | --- |
| EmployerName | Name of employer | This is to document who the employer is. |
| FreelancerName | Name of freelancer | This is to document who the freelancer is. |
| PayableAmount | Amount to be paid | This is used to keep track of the payment. |
| PaidAmount | Amount paid | This is used to keep track of how much is paid in this iteration and how much is remaining |
| DueAmount | Amount remaining | This is used to keep track of how much money needs to be paid after this iteration |
| ContractId | Link to the contract | This is used to link the contract with the payment details. |
| BillingDetails | Billing details of freelancer | This is needed to send over the payment to the freelancer. |

**Structural Database Rules (REVISED):**

To put down the structural rules for my database Artistree, let us go over my use cases from above and try to find out the entities and the relationships between them.

1. **Creating an account as an employer or a freelancer (REVISED):**

Here, we can see that there is an entity called User. This entity can be divided into two other types of entities, Employer and Freelancer. Every user when creating an account must identify themselves as either an Employer or a Freelancer and cannot be otherwise. So we can summarise the rule as follows:

1. Each user is an Employer or a Freelancer.
2. **Creating skills (REVISED):**

We can identify Skill and Operational as the 3rd and 4th entities respectively. The Skill entity is a large domain of skills that any user who is a freelancer may have. Skill has a relationship with User because a freelancer may have multiple skills and a skill may be had by multiple freelancers. It might happen that a skill can be created but a freelancer might not add it to their profile. By maintaining an entity called Operational, we will be able to break down the relationship described in the previous sentences. Here, in the entity Operational, we will find a set of data of skills that were added by freelancers to their profile to be identified by employers in the application. In this entity, a freelancer will always have at least one operational skill added to their profile and one operational skill has to be added (otherwise it will not be deemed operational) by one freelancer once only. On the other hand, a skill will be operational only when it is added by freelancers. So by that logic, each skill has to be operational for at least one freelancer to be operational. And each operational will always be addressing one skill only because a freelancer will only be able to choose that skill once. This can be summarised in the rule here:

1. A user **must** have **at least one** operational and each operational **must** be possessed by **one** user.
2. A skill **must** represent **at least one** operational and each operational **must** be representing **only one** skill.

No other relationships can be made between the 3 entities so we will move on to the 3rd use case:

1. **Creating Job/Project (REVISED):**

From here, we can consider Project to be another entity. A user who is an employer will be able to create one or many projects and assign the same or different users who are freelancers to each of them. It may also happen that an employer may not have created any project. Similarly, a freelancer may choose to do one or many projects. They may also not have chosen to do any, at a given point of time. But a project will always have to be created by an employer and each project will have one employer only. On the other hand, a project will have to be assigned to one freelancer only. But there can be a point of time when a project is open and it does not have any freelancer assigned to complete it. We can, hence, write down the following rules:

1. Each user **may** create **zero to many** projects.
2. Each project **has to be** worked upon by **at least one** user.

We can move on to the following use case, which is:

1. **Create Contract (REVISED):**

Here, we can identify Contract as another entity that seems to have a relation with the Project entity. A project can be the basis of zero to many contracts, in the event that a contract between an employer and freelancer expires and they might have to create a new one to establish the new agreement, or it can happen that a project might not have yet been made into a contract. But when a contract exists, it needs to have the project mentioned for which it is being made in the first place. And a contract will always identify one project. From this, we can come up with the following rules:

1. A project **may** be a part of **none to many** contracts.
2. Each contract **must** include **only one** project.

The final use case is given below:

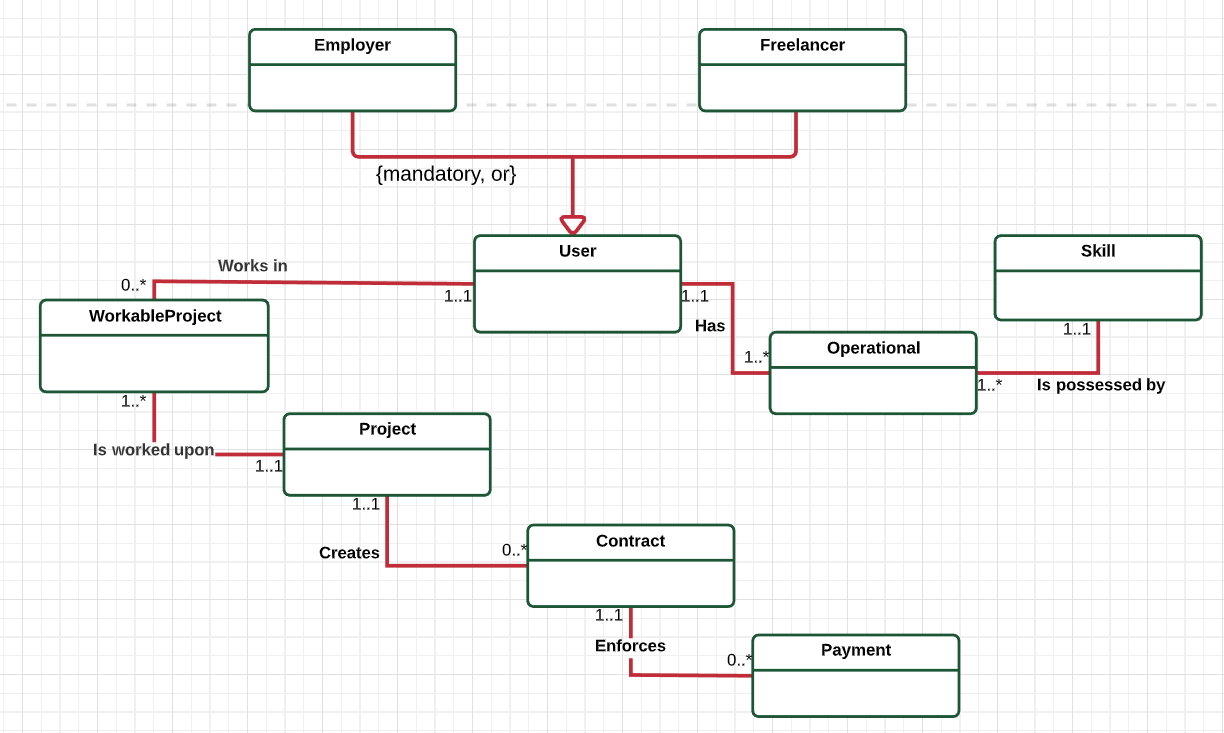
1. **Pay freelancer - Use Case (REVISED):**

The final entity is Payment, which has a relationship with Contract. A contract may have zero payments because there may be a time when no payment has yet been made for a contract, and the same contract may have multiple payments if the employer/freelancer have agreed to make transactions in installments. But a payment will always be under one contract. From here, we find the following rules:

1. A contract **may** generate **zero to many** payments.
2. A payment **must** always be enforced for **one** contract.

**Conceptual Entity‐relationship diagram (REVISED):**

The entity-relationship diagram that represents all of the rules above including entities: User, Employer, Freelancer, Skill, Operational, Project, Contract, and Payment is shown below:

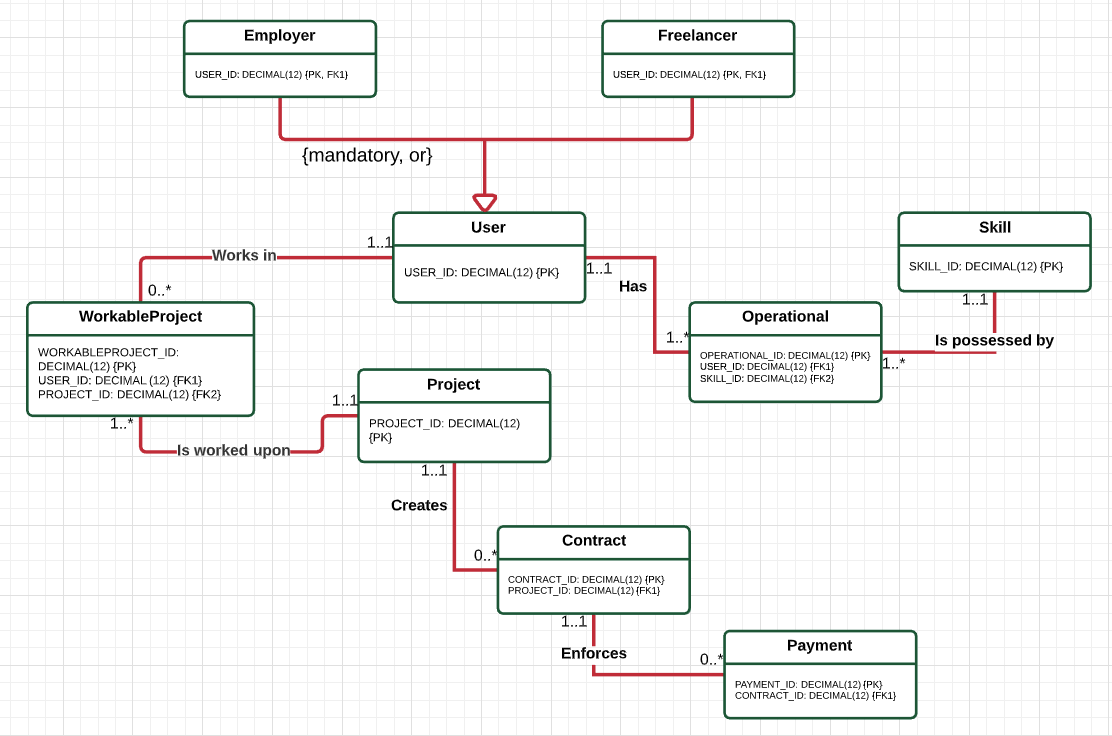


This ERD was made using LucidChart. I have added a supertype User which acts as a parent to subtypes Employer and Freelancer. The relationship, as described in rule#1, is representing a disjoint and totally complete constraint. This is shown by the {mandatory, or} text.

Also, the relationship between User and Project is a M:N project so I have created a bridge entity called WorkableProject that has 1:M relationships with both User and Project.

**Initial DBMS Physical ERD:**

The initial DBMS ERD is given below:



1. **Associative Relationships:**

All the relationships here are 1:M. So, I placed the Primary Key of the one side of the relationship as a foreign key in the many side of the relationship. I have used surrogate keys in all the tables. There was a M:N relationship between User and Project, but I had disintegrated it into two 1:M relationships, one between User and WorkableProject, and the second between Project and WorkableProject. The primary keys from User and Project have been placed as foreign keys in the WorkableProject table.

1. **Specialization-Generalization Relationships:**

There exists just 1 relationship like that and that totally complete disjoint relationship exists between supertype User and subtypes Employer and Freelancer. As the rules go, the primary key of User is inherited by both Employer and Freelancer as both Primary and Foreign keys.

**Summary and Reflection:**

In this iteration, I revised the business rules from last week. Every relationship rule has now been addressed in one sentence. I have also created a new entity called User that acts as a supertype for previously acknowledged entities: Employer and Freelancer. I have accommodated this entity into the Conceptual ERD. I dropped the previously stated rules between Employer and Freelancer tables with the Project table. Instead, I made a M:N relationship between User and Project and created a bridge entity to connect the two. This can be seen in business rule#3. I also revised business rule#2 to create the relationship of User with Operational and Project, instead of having it with Freelancer. Finally I have created an initial DBMS Physical ERD which accommodates the new relationships along with the constraints for each table.

I was worried how I would accommodate the supertype in a complex relationship with the Project table, but I hope I can move forward with the design I have struck upon here.