QUICKGIG

Opportunities Ahead

By:

Mathew Migliore, Peter Rudolf, Tyler Segovia, and Kevin Romero

Presented to:

DURHAM COLLEGE
SUCCESS MATTERS

Date:

2021-02-13

TABLE OF CONTENTS

INTRODUCTION

QuickGig is an app that allows users to post and search for job requests and make bids on small jobs. Users can post a picture and specifications of the job they want and make bids on the job based on how much they are willing to do the job for. The employer will choose the best bid. Employers will be able to allow their gig to been seen on the map and whether users can make bids on that job. The app will also notify you about jobs based on your profile and jobs that are posted that you would be interested in. Transactions done through the app will work with the employer puts down their balance, and when the job is completed, the worker will post a photo of the job and request the money. Both the employer and the employee will have a layer of security for them to get paid when the job is complete, or the employer to ensure that their money is not stolen if the job is not done, or something has been done incorrectly.

MISSION STATEMENT

The purpose of QuickGig is to manage and promote job listings to simplify the process of finding a job for anyone.

OBJECTIVES

- To maintain (enter, update, delete) data on users
- To maintain (enter, update, delete) data on posts
- To maintain (enter, update, delete) data on jobs

- To perform searches on users
- To perform searches on posts
- To perform searches on jobs

- To track the status of users
- To track the status of posts
- To track the status of jobs

- To report on users
- To report on posts
- To report on jobs

USER VIEW PREFACE

For the user views we took into consideration the employees that would access the system and what each employee should or should not be able to see and/or do while logged in. We managed to break the system down into 6 categories ranging from low to full access of the system.

LOW ACCESS:

This is a split between Kitchen Staff, Room Cleaning, Accounting & Stock, and Front Desk. They are each able to view, update, and edit certain parts of the Users information, but are not able to do everything for the users. They are also unable to edit the information for themselves or any other higher-level user.

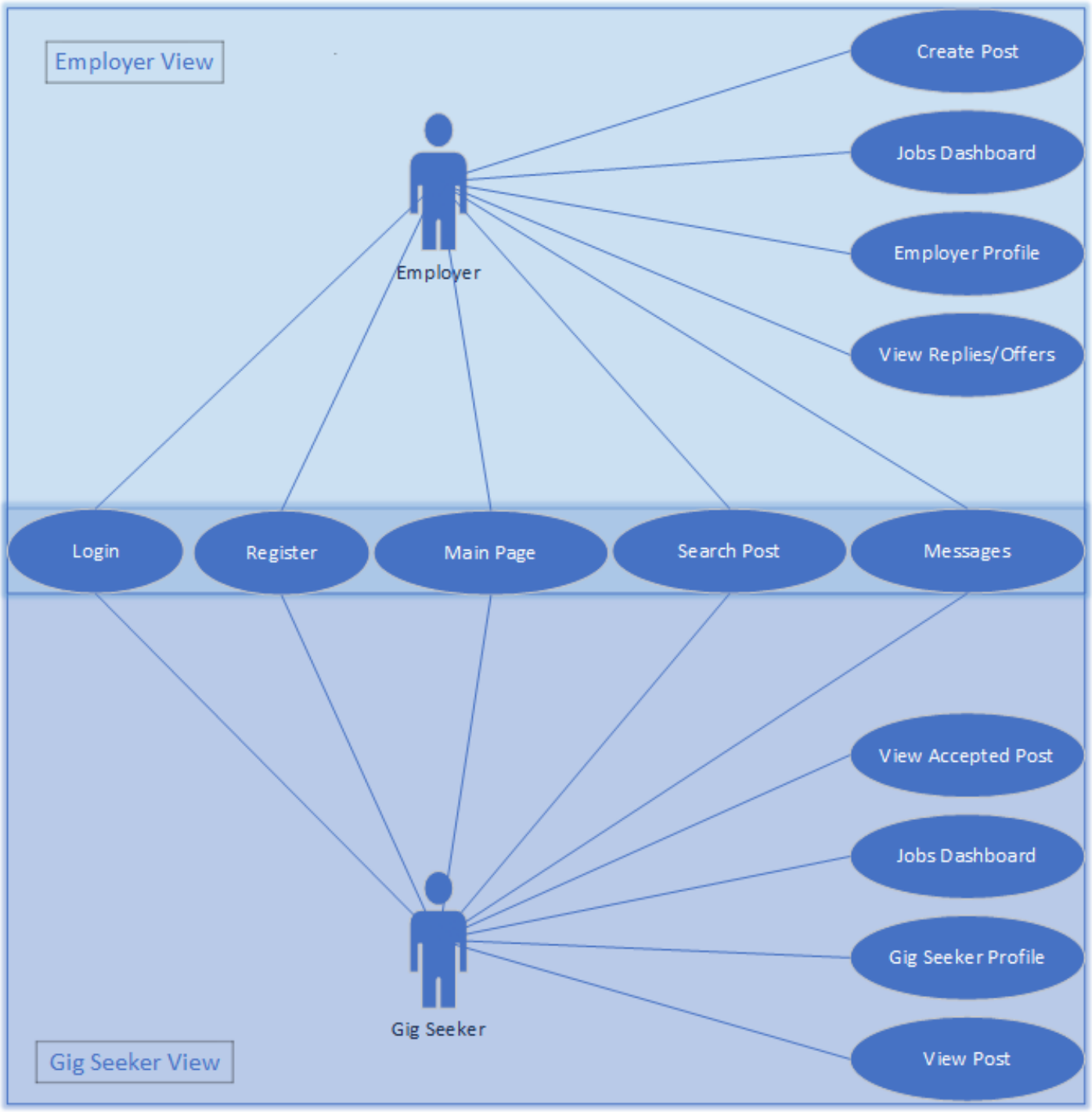
HIGH ACCESS:

Management has access over all the previous users, can edit, update, and the records, but do not have full access. They cannot delete records.

FULL ACCESS:

Admin has full access over everything. They can edit, update, and delete all records and all users.

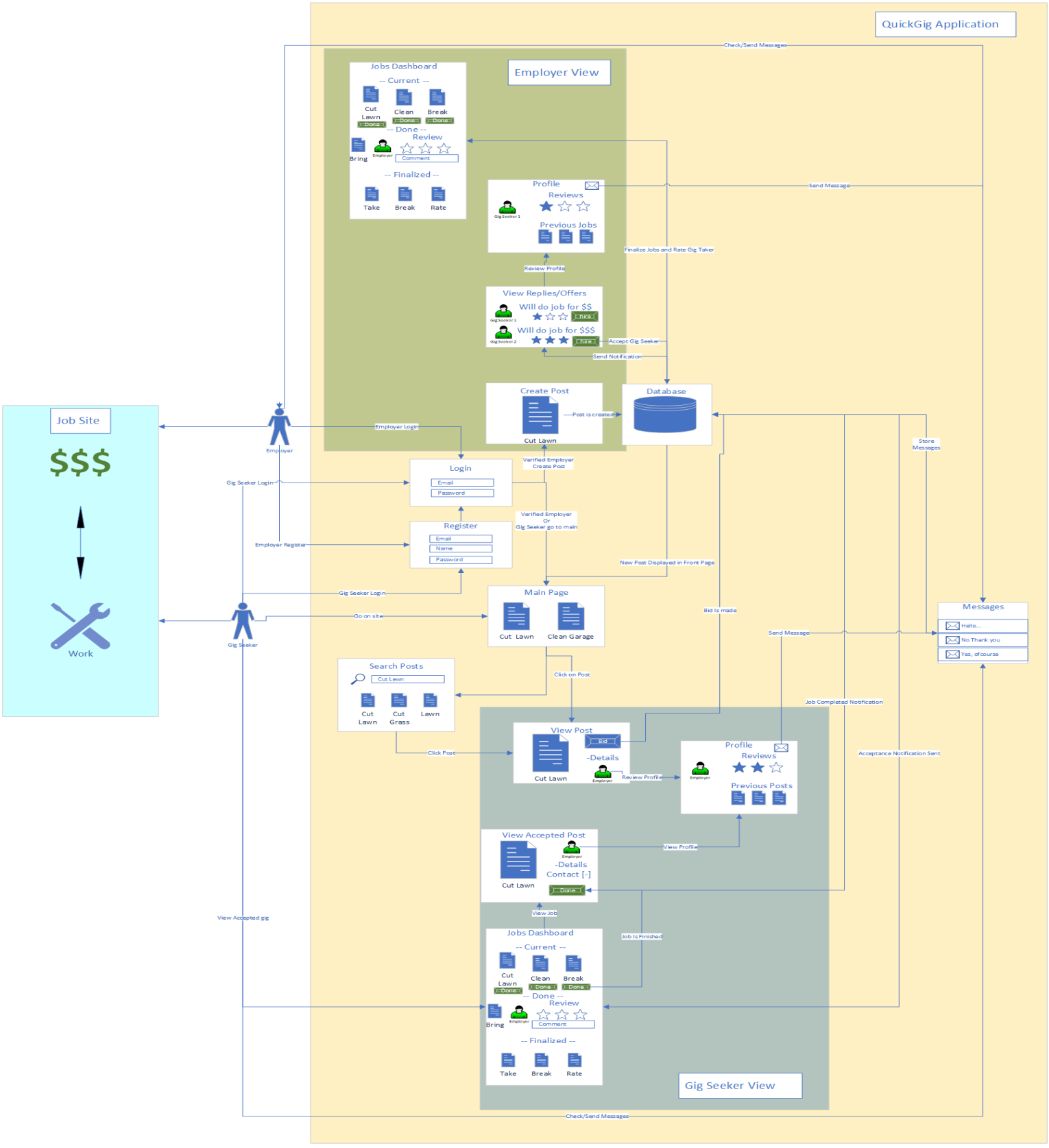
Use Case Diagram



SYSTEM BOUNDARY PREFACE

For the System Boundary we've took a lot of inspiration from the user view and mostly wanted to focus on the six major groups and their basic operations within the system that they would go through when creating, viewing, and modifying any information from the system. Once we had that we thought about other external groups that interacted with those functions.

SYSTEM BOUNDARY



3NF Preface

Introduction: For the data normalization our group took a different approach than the standard, instead of starting with the list, we wanted to create the graph so we could envision how the data would really work within our heads, and how we could organize it in a way that it was programmatically easy to insert, create and update tables. Once the graph was completed, we translated that to the regular 3NF list, which helped us find more logical error we missed in the graph.

How did we come up with the design?

We began the design with the most important table, which was posts. We tried to envision how the entire system came together from the time a user is created, to the moment they finish a job. We then visualized how a user would interact with our system and what functionality they would need.

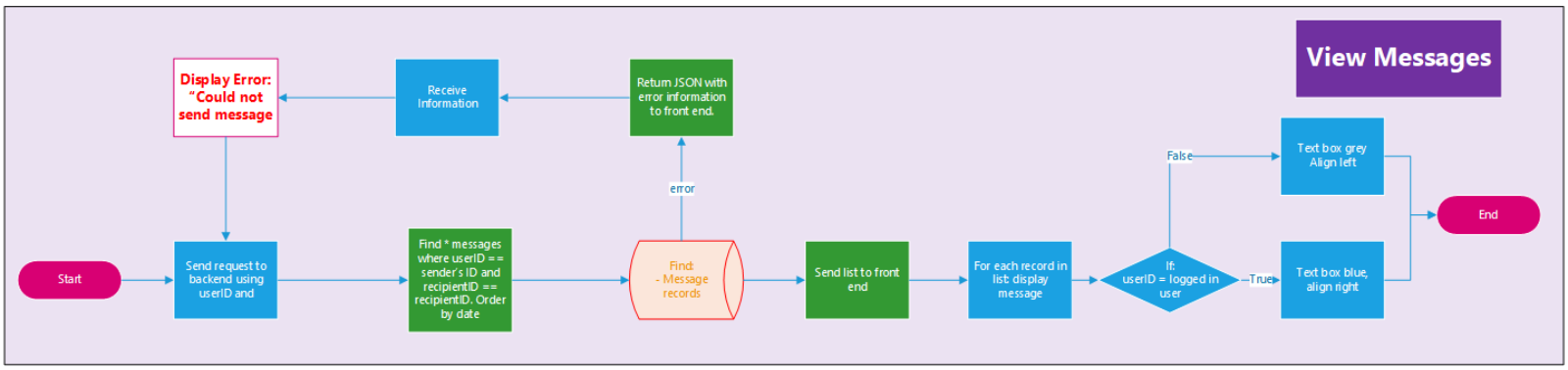
What were the biggest challenges?

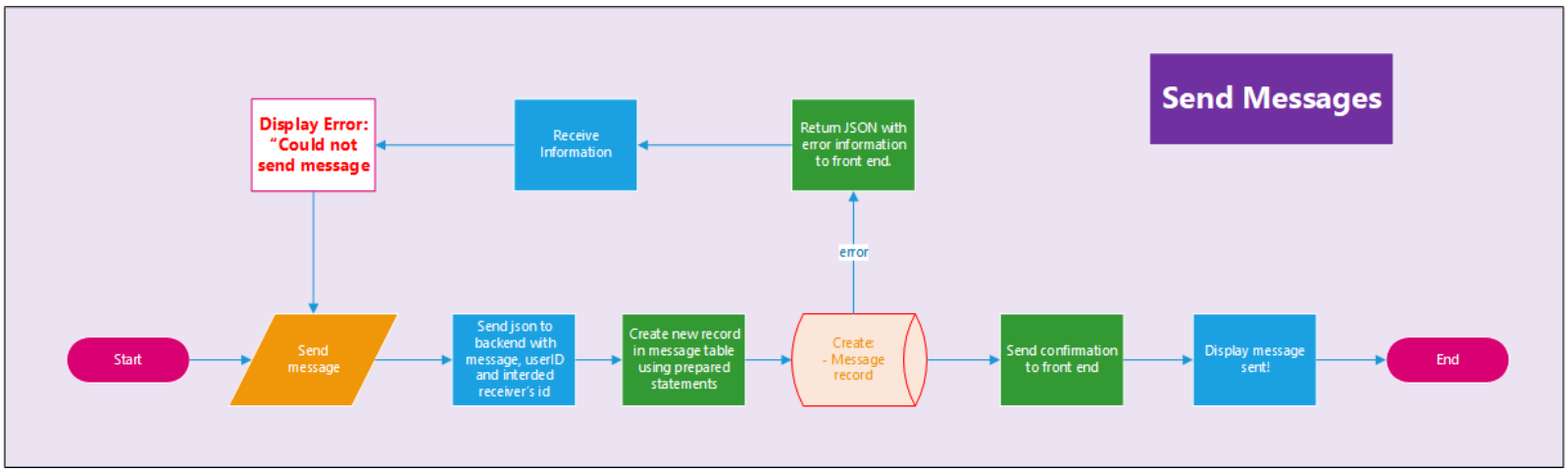
The biggest challenges were thinking how we could fulfil every need of someone looking for a job and someone listing a job, without going out of scope.

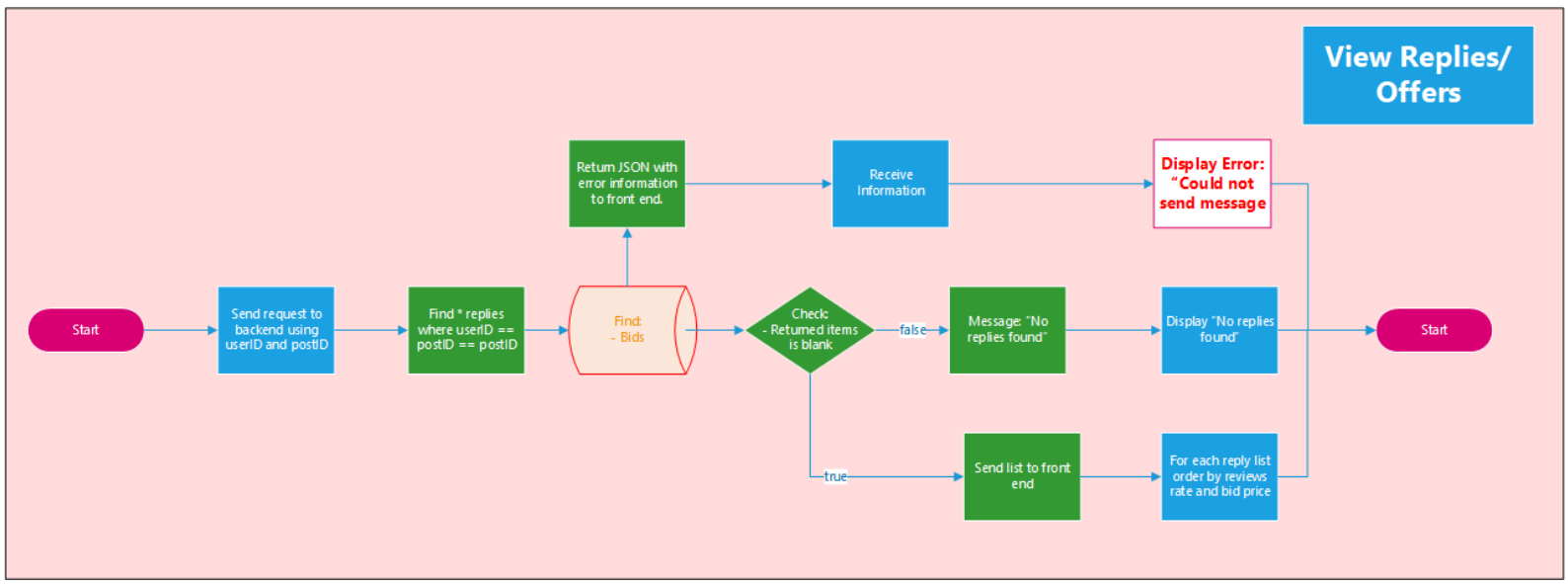
Why did our group choose those tables?

It is our belief that these tables store all the necessary data to perform the functions required and also allow the development team to expand their system for future endeavors.

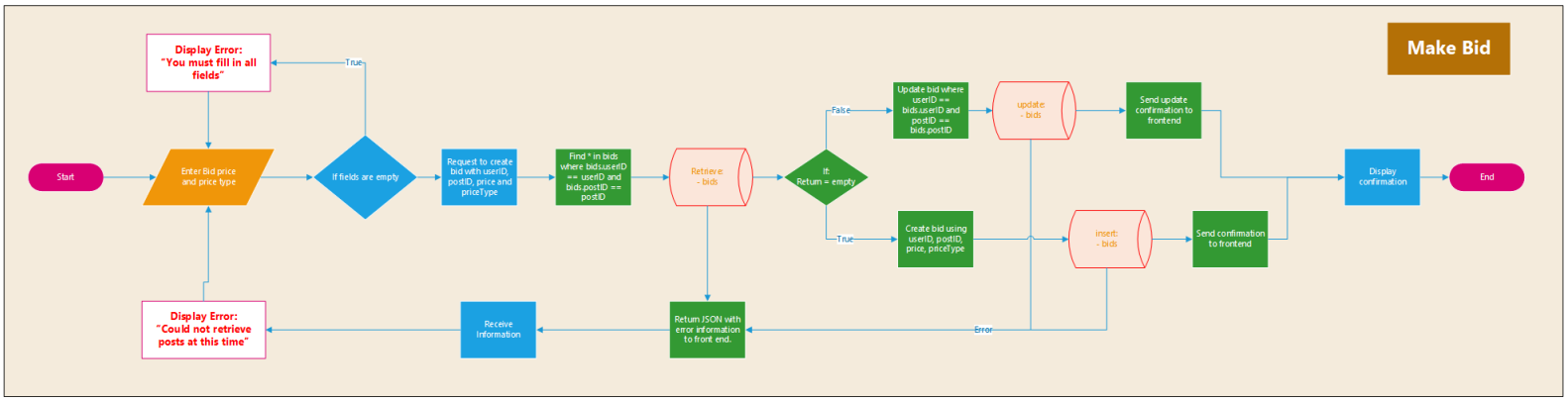
Activity Diagrams

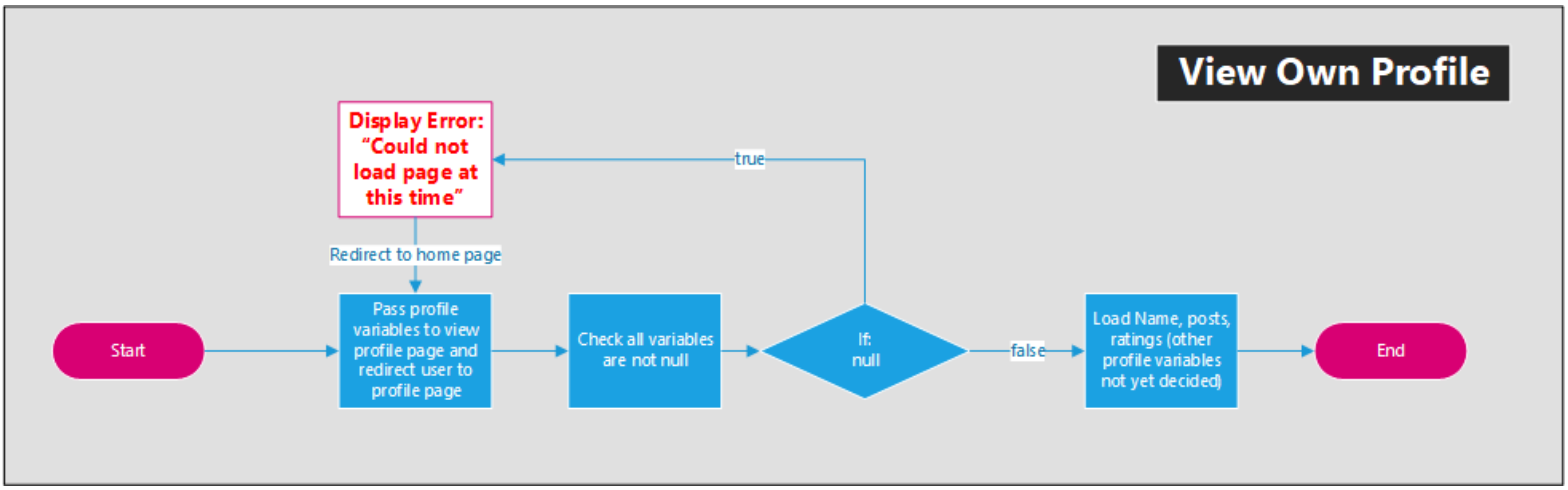


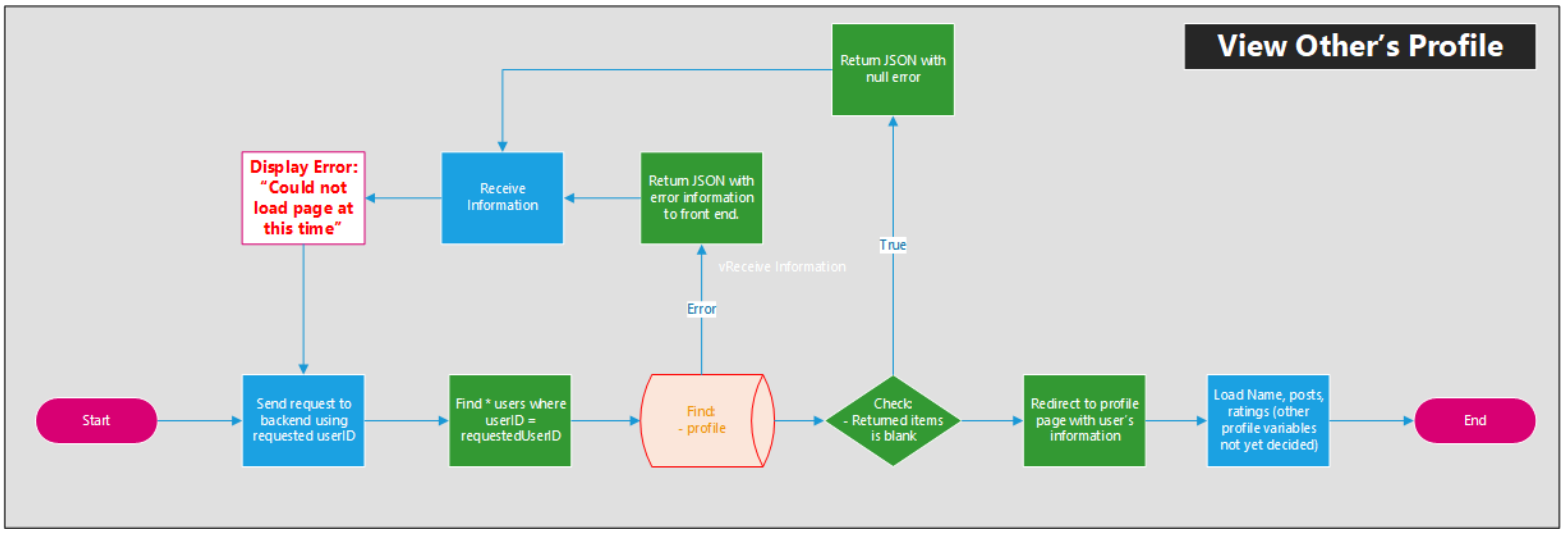




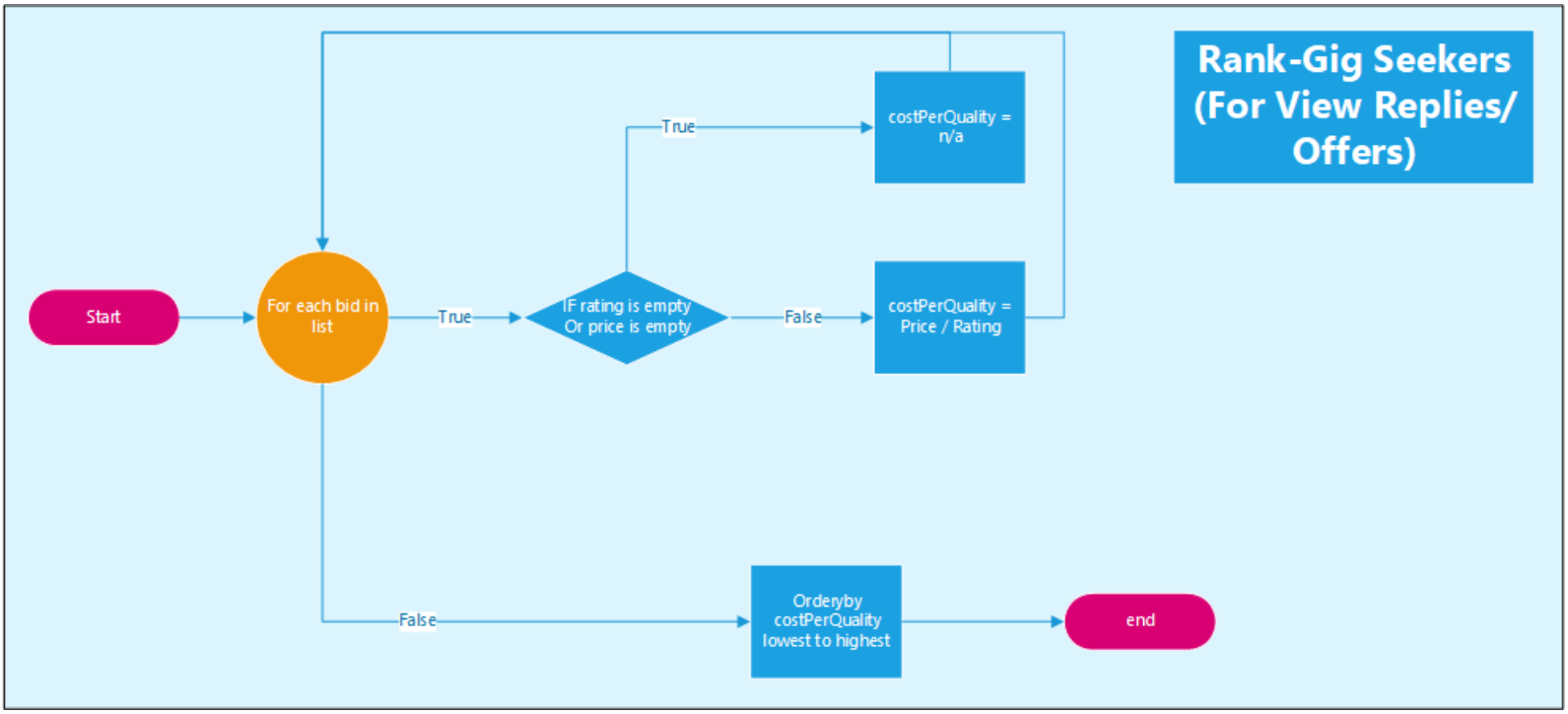
Activity Diagrams

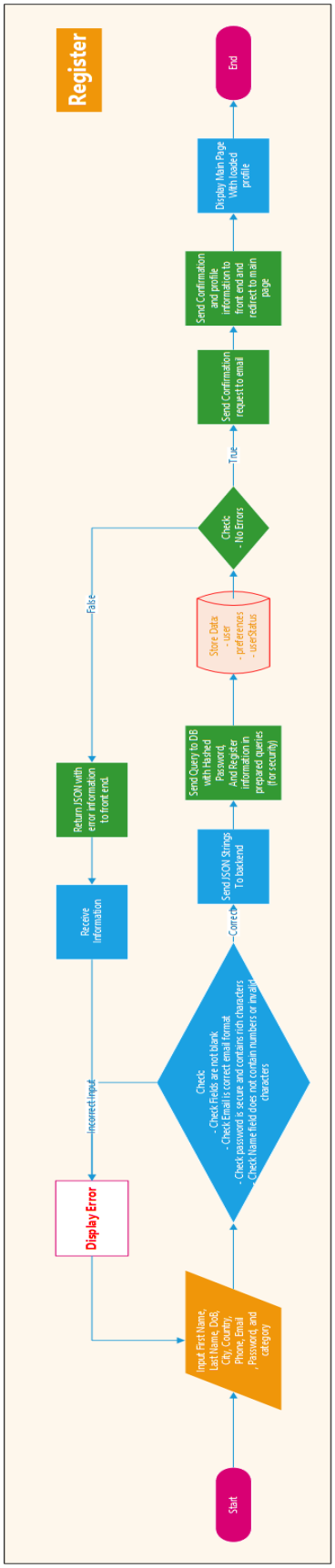






Activity Diagrams




Register
Start
Input First Name, Last Name, DOB, City, Country, Phone, Email , Password, and category
Check:
- Check Fields are not blank
- Check Email is correct email format
- Check password is secure and contains rich characters
- Check Name field does not contain numbers or invalid characters
Incorrect Input
Display Error
Receive information
Return JSON with error information to front end.
False
Correct
Send JSON Strings To backend
Send Query to DB with Hashed Password, And Register information in prepared queries (for security)
Store Data
- user
- preferences
- userStatus
Check:
- No Errors
True
Send Confirmation request to email
Send Confirmation and profile information to front end and redirect to main page
Display Main Page With loaded profile
End

3NF Preface

Introduction: For the data normalization our group took a different approach than the standard, instead of starting with the list, we wanted to create the graph so we could envision how the data would really work within our heads, and how we could organize it in a way that it was programmatically easy to insert, create and update tables. Once the graph was completed, we translated that to the regular 3NF list, which helped us find more logical error we missed in the graph.

How did we come up with the design?

We began the design with the most important table, which was posts. We tried to envision how the entire system came together from the time a user is created, to the moment they finish a job. We then visualized how a user would interact with our system and what functionality they would need.

What were the biggest challenges?

The biggest challenges were thinking how we could fulfil every need of someone looking for a job and someone listing a job, without going out of scope.

Why did our group choose those tables?

It is our belief that these tables store all the necessary data to perform the functions required and also allow the development team to expand their system for future endeavors.

Legend

Table Name (Primary Key (PK), Foreign Key (FK), Attribute Name…**)**

Profile:

users(**userID (PK)**, **addressID (FK)**, **userStatusID (FK),** email, password, firstName, lastName, phone, dateOfBirth, city, country, bio, preferencesIndex)

favourites(**favouriteID (PK), postID (FK), userID (FK)**, active, date)

userStatus(**userStatusID (PK)**, status)

userPhotos(**photoID (PK), userID (FK)**)

loginHistory(**loginID (PK)**, **userID (FK)**, date)

Posts:

posts(**postID (PK)** , **userID (FK)**, **addressID (FK)**, **postStatusID (FK)**, **catergoryID (FK),** postTitle, postDescription, setPrice, budget, postedDate, visible) optional: (views)

contract(**contractID (PK), userID (FK), postID (FK), contractStatusID (FK**), dateStart, dateEnd)

postStatus(**postStatusID (PK)**) (open, accepted, complete, deleted)

contractStatus(**contractStatusID (PK)**) (open, hidden, accepted, disputed, done, finalized, deleted)

postPhotos(**photoID (PK), postID (FK)**)

contractCategories(**categoryID (PK),** categoryName, categoryDescription)

bids(**bidID (PK), postID (FK), userID (FK)**, price, priceType, **bidStatusID (FK)**, createdDate, responseDate,)

bidStatus(**bidStatusID (PK),** status)

priceType(**priceTypeID (PK),** priceType)

Reviews:

review(**reviewID (PK), contractID(fk), userID(fk), reviewerID(PK)**, comment, rating)

Messages:

messages (**messageID (PK), senderID(fk), sendToID(fk),** title, message)

Others:

addresses(**addressID (PK)**, line1, line2, city, postalCode, province, country)

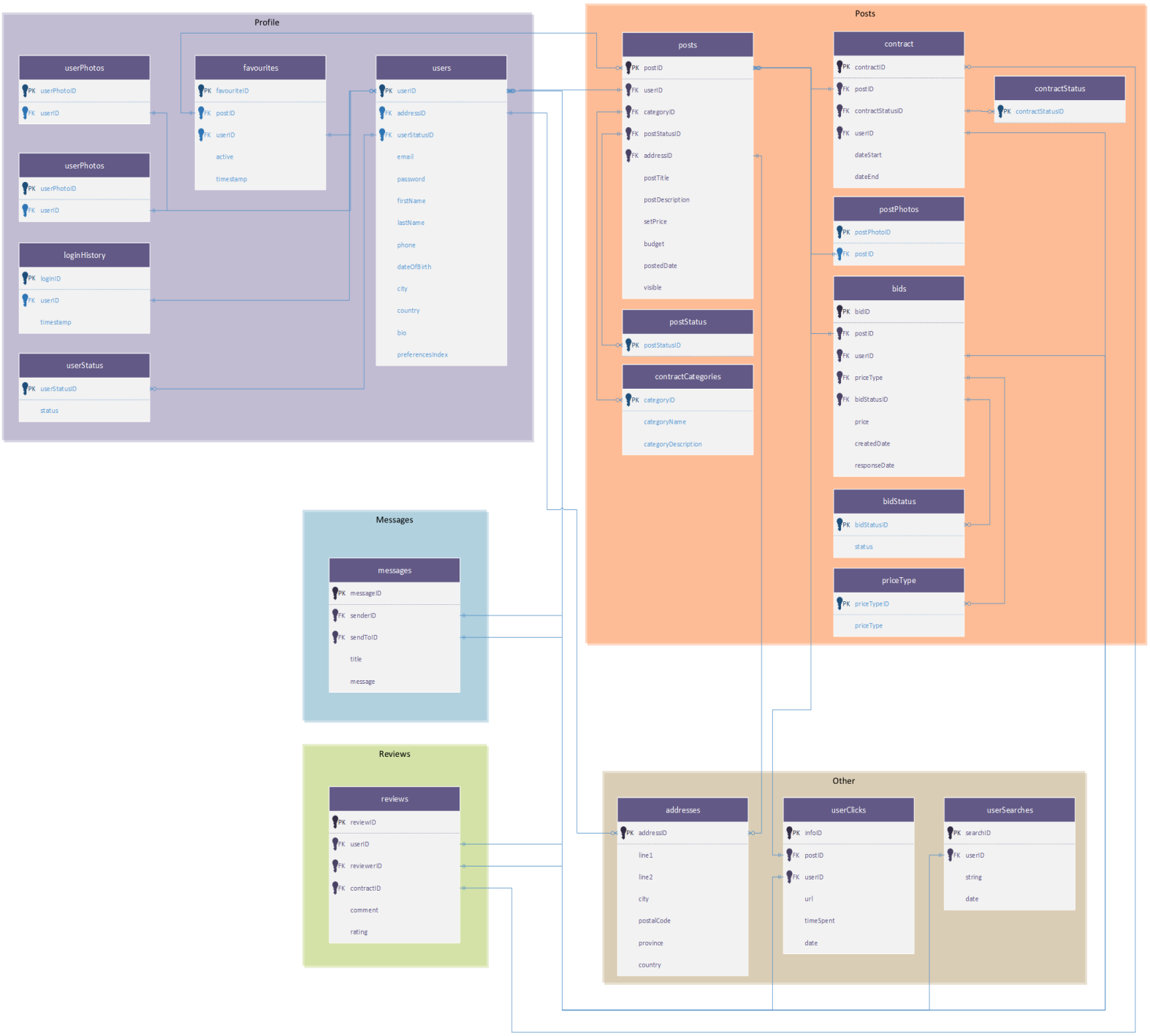
userClicks(**infoID(PK)**, **userID(fk)**, **postID(fk)**,url, timeSpent,date)

userSearches(**searchID(PK), userID(fk)**, string, date**)**

Extra:

- ~~Track what they're searching for~~
- ~~Track the jobs the user bids on~~
- ~~Track time spent on page~~
- ~~Create personalized suggestions~~
- ~~Track number of clicks/visits a post gets~~
- ~~Hide reviews~~

3NF DIAGRAM


Profile
userPhotos
PK userPhotoID
FK userID
favourites
PK favouriteID
FK postID
FK userID
active
timestamp
users
PK userID
FK addressID
FK userStatusID
email
password
firstName
lastName
phone
dateOfBirth
city
country
bio
preferencesIndex
userPhotos
PK userPhotoID
FK userID
loginHistory
PK loginID
FK userID
timestamp
userStatus
PK userStatusID
status
Posts
posts
PK postID
FK userID
FK categoryID
FK postStatusID
FK addressID
postTitle
postDescription
setPrice
budget
postedDate
visible
contract
PK contractID
FK postID
FK contractStatusID
FK userID
dateStart
dateEnd
contractStatus
PK contractStatusID
postPhotos
PK postPhotoID
FK postID
bids
PK bidID
FK postID
FK userID
FK priceType
FK bidStatusID
price
createdDate
responseDate
postStatus
PK postStatusID
contractCategories
PK categoryID
categoryName
categoryDescription
bidStatus
PK bidStatusID
status
priceType
PK priceTypeID
priceType
Messages
messages
PK messageID
FK senderID
FK sendToID
title
message
Reviews
reviews
PK reviewID
FK userID
FK reviewerID
FK contractID
comment
rating
Other
addresses
PK addressID
line1
line2
city
postalCode
province
country
userClicks
PK infoID
FK postID
FK userID
url
timeSpent
date
userSearches
PK searchID
FK userID
string
date

Functional Dependencies Preface

Introduction: For the functional dependencies we thought about which attributes would logically determine the others within the database, this is where the graph we made for the 3NF design came in handy, we were able to visualize within a table each attribute's role within the database.

How did we come up with the design?

The standard Functional Dependency format was followed.

What were the biggest challenges?

The biggest challenges laid in determining which attributes in the table would allow us to search for other attributes and which ones wouldn't.

Legend

Table Name:

{Determining Attribute/s (X)} -> {Functionally Dependent Attribute/s (Y)}

Functional Dependencies

users:

{userID} -> {addressID, userStatusID, email, password, firstName, lastName, phone, dateOfBirth, city, country, bio, prefrencesIndex}

{phone, email} -> {userID, addressID, userStatusID, password, firstName, lastName , dateOfBirth, city, country, bio, prefrencesIndex}

{fistName, lastName, email} -> {userID, addressID, userStatusID, password, dateOfBirth, city, country, bio, prefrencesIndex}

{dateOfBirth, email } ->{userID, addressID, userStatusID, password, firstName, lastName , city, phone, country, bio, prefrencesIndex}

{ fistName, lastName, addressID} -> {userID, userStatusID, password,email, phone, dateOfBirth, city, country, bio, prefrencesIndex}

favourites:

{favouriteID}-> {**postID, userID**, active, date}

{postID, userID}-> {favouriteID, active, date}

{userID}->{favouriteID, postID, active, date}

userStatus:

userPhotos:

loginStatus:

posts:

{postID}->{userID, addressID, postStatusID, categoryID, postTitle, postDescription, setPrice, budget, postedDate, visible}

{postTitle, categoryID}->{postID, userID, addressID, postStatusID, categoryID, postTitle, postDesecription, setPrice, budget, postedDate, visible}

contract:

{contractID, userID}->{postID, contractStatusID, startDate, endDate}

postStatus:

contractStatus:

{contractStatusID}->{open, hidden, accepted, disputed, done, finalized, deleted}

postPhotos:

contractCategories:

{categoryID}->{categoryName, categoryDescription}

bids:

bidID (PK), postID (FK), userID (FK), price, priceType, bidStatusID (FK), createdDate, responseDate,)

{bidID, postID, userID, bidStatusID}->{price, priceType, createdDate, responseDate}

bidStatus:

priceType:

review:

{reviewID}->{contractID, userID, reviewerID, comment, rating}

messages:

{messageID, senderID, sendToID}->{title, message}

addresses:

{addressID}->{line1, line2, city, postalCode, province, country}

userClicks:

Data Dictionary Preface

Introduction: For the data dictionary, we spend quite a bit of time looking at the things it was being asked of us again in the system request. We really wanted to make sure that the sizes of data were consistent with the report that were presented and the other possible type of data that could be stored. We tried to have a balance with the storage and the user friendliness when entering data. One of the reasons we took the graphical approach for the 3NF design was because we knew the immense help it would provide use for developing the Data Dictionary. With the combination of the 3NF design, functional dependencies, and this data dictionary, we were able to catch errors it the original designs and correct them before it was handed over to the development team.

DATA DICTIONARY

Table	Column	Data Type	References	Default	Not Null
< users >	email	VARCHAR		Identity	Y
< users >	addressID	VARCHAR	<addresses>.addressID		Y
< users >	userStatusID	VARCHAR	<userStatus>.userStatusID		Y
< users >	password	VARCHAR			Y
< users >	firstName	VARCHAR			Y
< users >	lastName	VARCHAR			Y
< users >	phone	VARCHAR			Y
< posts >	postID	INT		Identity	Y
< posts >	addressID	INT	<addresses>.addressID		
< posts >	postStatusID	INT	<postStatus>.postStatusID		
< posts >	postTitle	VARCHAR			Y
< posts >	postDescription	VARCHAR			Y
< posts >	setPrice	MONEY			Y
< posts >	minPrice	MONEY			N
< posts >	maxPrice	MONEY			N
<customers>	customerID	INT		Identity	Y
<customers>	firstName	VARCHAR			Y
<customers>	lastName	VARCHAR			Y
<customers>	phoneNumber	VARCHAR			Y
<customers>	addressID	INT	<addresses>.addressID		Y
<addresses>	addressID	INT		Identity	Y
<addresses>	addressLine1	VARCHAR			Y
<addresses>	addressLine2	VARCHAR			N
<addresses>	city	VARCHAR			Y
<addresses>	provinceID	INT	<provinces>.provinceID		Y
<addresses>	country	VARCHAR		'Canada'	Y
<addresses>	postalCode	CHAR			Y
<provinces>	provinceID	INT		Identity	
<provinces>	provinceCode	CHAR			Y
<provinces>	provinceName	VARCHAR			Y
<customerBilling>	billID	INT		Identity	Y
<customerBilling>	billAmount	MONEY			Y
<customerBilling>	reservationID	INT	<reservations>.reservationID		Y
<customerBilling>	paymentType	VARCHAR			Y
<customerBilling>	amountOwing	MONEY			Y
<chargeableItems>	itemID	INT		Identity	Y
<chargeableItems>	itemName	VARCHAR			Y
<chargeableItems>	itemDescription	VARCHAR			Y
<chargeableItems>	itemPrice	MONEY			Y
<transactions>	transactionID	INT		Identity	Y
<transactions>	billID	INT	<customerBilling>.billID		Y

<transactions>	itemID	INT	<chargeableItems>.itemID		Y
<transactions>	amountOfItems	INT			Y
<transactions>	date	DATETIME			Y
<staff>	staffID	INT		Identity	Y
<staff>	firstName	VARCHAR			Y
<staff>	lastName	VARCHAR			Y
<staff>	phoneNumber	VARCHAR			Y
<staff>	addressID	INT	<addresses>.addressID		Y
<staff>	positionID	INT	<employmentPositions>.positionID		Y
<staff>	salary	MONEY			Y
<staff>	hiredDate	DATE			Y
<staff>	terminationDate	DATE			N
<staff>	photo	VARCHAR			N
<employmentPositions>	positionID	INT		Identity	Y
<employmentPositions>	positionTitle	VARCHAR			Y
<employmentPositions>	positionDescription	VARCHAR			Y
<employmentPositions>	startingSalary	MONEY			Y
<users>	userID	INT		Identity	Y
<users>	username	VARCHAR			Y
<users>	password	VARCHAR			Y
<users>	roleID	INT	<roles>.roleID		Y
<users>	staffID	INT	<staff>.staffID		N
<roles>	roleID	INT		Identity	Y
<roles>	roleTitle	VARCHAR			Y
<roles>	roleDescription	VARCHAR			Y

FINAL WORD

We hope that the solutions that we have created above are exactly to your specifications and likings. We have taken into consideration every piece of information you have given us and concluded that our research and findings will provide you with an excellent application and database. If you do find anything you would like to change about our report or the application, please do not hesitate to contact us. Do not hesitate to provide any feedback on any of the information we have provided here today.

Thank you for choosing us, and we hope to hear back from you soon.

Best Regards,

Mathew Migliore, Tyle Segovia, & Kevin Romero.

Dictionary

- **DBMS:** Short for Database Management System. The system that manages the storage and querying of a database.

- **Query:** A logical request made to the database to return specified results.

- **Mission Statement:** Defines an understanding of what the mission of the project is and what the company wants to accomplish from this project.

- **Mission Objectives:** Defines in a simple format what the objectives of our team are for development in order to deem the project successfully completed.

- **System Boundary:** Defines the boundaries that are internally in the facility and externally. It also defines the actors, items and other systems that interact with the company's system.

- **Use Case:** Defines the objects and functions that the actors within the system interact with in each systematic cycle.

- **3NF:** Third Normal form, is the most common used normalization form to design a database which uses principles to reduce the duplication of data, avoid data anomalies, ensure referential integrity, and simplify data management.

- **Functional Dependencies:** defines a relationship between two attributes, typically between the PK and other non-key attributes within a table.

- **Data Dictionary:** a set of information describing the contents, format, and structure of a database and the relationship between its elements, used to control access to and manipulation of the database.

- **Attribute:** In terms of databases it refers to as a field. Used interchangeably with the term "column"

- **Column: :** In terms of databases it refers to as a field. Used interchangeably with the term "Attribute"

- **Primary Key:** The field used to Identify a record.

- **Foreign Key:** The field that reference the primary key of another table.

Disclaimer

Quickgig is a fictional company for the computer programmer and analysis program's capstone project for Durham College. Use of names, logos, references, and resources are for learning purposes only. All resources, names, logos, slogans, etc. were created by the hardworking students in the "Quickgig" Project, any similarities to external sources is purely coincidental.