Library Media Rental Software

Jeffrey Van (jeffrey.van@sjsu.edu, 013941317)

Jose Velasco (jose.velascojuarez@sjsu.edu, 016184753)

Dawson Williams(dawson.williams@sjsu.edu, 012135071)

Description:

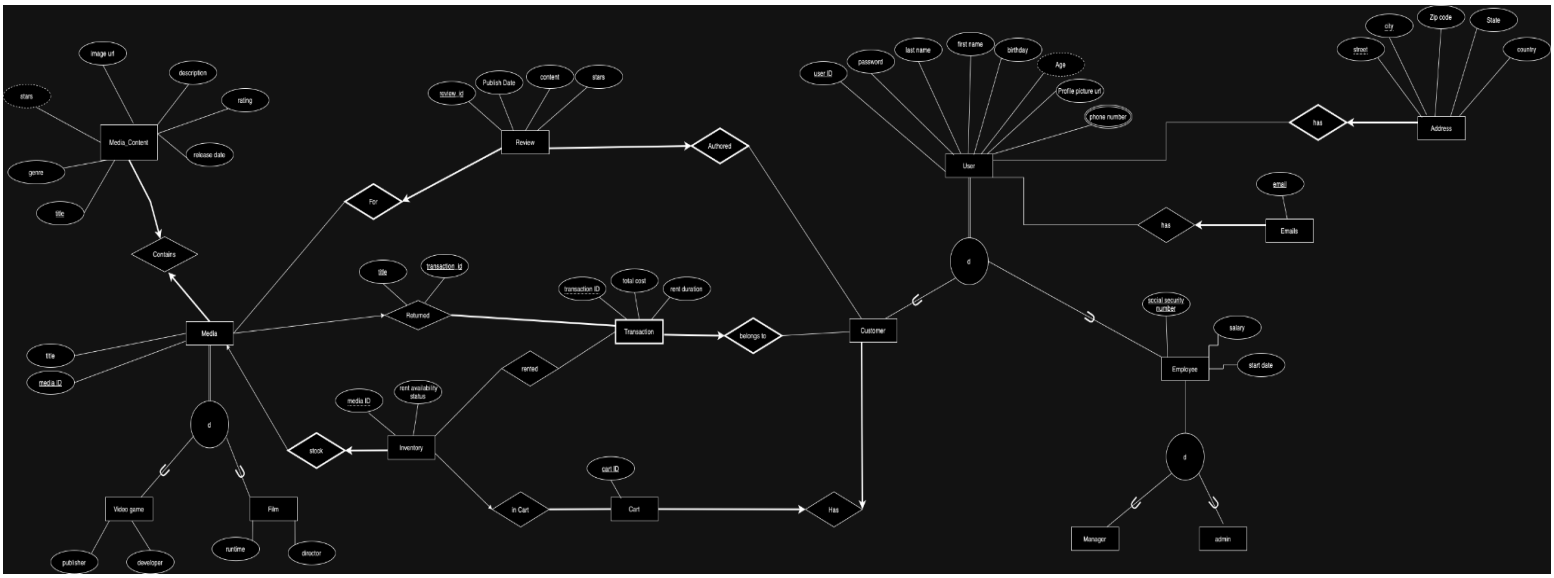
The software we created is a simulation of a library media rental business. The business stocks two different forms of digital media being movies and video games. The business keeps track of rentals through various transactions. The software keeps track of every customer through an account they make before being able to interact with it. Additionally, every employee is tracked as well. Our goal with this project is to mimic a system akin to blockbuster, but in a digital medium. You are able to make an account, browse inventory, and take out a rental with one of the forms of media that are currently stocked.

Architecture:

The software stack comprises multiple different technologies. For the backend, we utilized python. The backend uses pymysql as the library used to interact with the MySQL database containing all of the relevant tables in order to keep track of different elements of the software. Additionally, the back end utilizes FastAPI in order to create API endpoints for the front end to be able to communicate with. Within FastAPI, other technologies, such as Pydantic for type checking, and OAuth2 for security were implemented. For the front end of the stack, we utilized React. With React, we can use Axios to send HTTP requests to the endpoints made with FastAPI to display different forms of data that are kept within the MySQL database.

EER Diagram:

We utilized drawIO in order to create our EER diagram. A PNG of the diagram will be shown below. If a larger view is needed to see the diagram in more detail, it will be provided below.
Link to diagram for a bigger view:
https://app.diagrams.net/#G15xzkkGyNFw9UlAKIKlsYJomd7ct0A8No



Database Design:

Media (<u>media_id</u>, title)
FD1: media_id -> title,

Media_Content(<u>title</u>, genre, rent_price, image_url, description, release_date, rating)
FD1: title -> genre, description, release_date, image_url, rating

Video Game (<u>media_id</u>, publisher, developer)
FD1: media_id -> publisher, developer

Film (<u>media_id</u>, runtime, director)
FD1: media_id -> runtime, director

User (<u>user_id</u>, password, FirstName, LastName, Birthday, Profile_picture_url, phone_number)
FD1: user_id -> password, FirstName, LastName, Birthday, Profile_picture_url, phone_number

Address (<u>user_id, street, city</u>, zip_code, state, country)
FD1: user_id -> street, city, zip_code, state, country

Email (user_id, email)
FD1: user_id -> email
FD2: email -> user_id

Inventory (media_id, rent_availiabilty_status)
FD1: media_id -> rent_availiabilty_status

Transaction(transaction_id, user_id, total_cost, rent_duration)
FD1: transaction_id, user_id -> total_cost, rent_duration

Rented (transaction_id, media_id)
FD1: transaction_id -> media_id
FD2: media_id -> transaction_id

Review(review_id, user_id, media_id, publish_date,content, stars)
FD1:review_id, user_id, media_id -> publish_date,content, stars
FD2: review_id -> publish_date,content, stars
FD3: user_id, media_id -> review_id
FD4: review_id -> media_id, publish_date, content, stars
ReviewContent(review_id, media_id, publish_date, content, stars)
Reviews(review_id, user_id)

Cart (cart_id, user_id)
FD1: cart_id -> user_id
FD2: user_id -> cart_id

In cart(media_id, cart_id)
FD1: media_id -> cart_id
FD2: cart_id -> media_id

Returned(transaction_id, title)
FD1: transaction_id -> title

Design Decisions

Database:

At the beginning, we had numerous tables that were creating many redundancies for us. We decided to prune back the sheer amount of data that we stored within the database in order to give it more focus on our goals. An example of this was creating additional employee tables, one for Admin and one for Manager where the only thing that they provided were the title of the position, but nothing more. We decided to encapsulate that into the employee table using an enum to declare an employee an Admin, Manager, or None.

Backend:

We decided on using python for the backend language of choice for the simplicity in the syntax and the power rof the libraries it has to offer. We decided to utilize pymysql over other libraries, such as sqlalchemy as pymysql offers a low level interaction with MySQL as you write the SQL statements yourself instead of having an abstracted interaction with it. Additionally, we wanted to use FastAPI to create our API endpoints because, as the name implies, it is able to create the endpoints relatively fast and has other useful functions like security and pydantic support for type checking.

Front End:

We made the design decision of using TypeScript and React because TypeScript is a more powerful verison JS since it includes static typing which would produce better code and allow for more efficient bug handling. The React framework also allows us to be more efficient than using just pure HTML/CSS alone, the framework provides many useful components and libraries that we can take advantage of and manipulate to make our front end more visually appealing. On top of this, our front end tech stack also made it so that we could effectively run asynchronous methods in the background to gather information and take in information.

Implementation Details:

The way that we deal with MySQL is through pymysql on the backend of our application. We initially created DDL statements to create our tables and then made files for each table to handle their respective CRUD operations. For example, when you go to make a new customer, that logic is handled in the “user_dao.py” file. Within said file, there is a database manager that is imported that creates a connection to our MySQL database. Furthermore, there are different functions that deal with the CRUD operations that relate to a user. After creating the CRUD operations, we made a separate folder that contains the endpoints to our API with each entity. Using the same users example, there is a “users_route.py” file that is responsible for creating

such endpoints. The endpoints call the functions within the “user_dao.py” file so that the front end is able to directly access them.

For the front end we used TypeScript and React, and using axios, we linked the front end to send requests to the back end server to perform CRUD operations. We used axios to use the endpoints to access the database and the operations that we needed. We also had reusable components that were reused across many pages to allow for more reusable functionality, instead of constantly rewriting components. React use-context functionality was used to store the logged in user meta information such as user_id, access token, etc globally across the web application’s pages.

Demonstration of System Run:

https://youtu.be/u6si9bUWHIY

1. Create a new user
2. Log into user
3. Browse Media
4. Add to cart
5. Check out cart
6. View transactions
7. Return media

What we plan on doing next:

If we were to take this project further, we would want to expand the amount and type of inventory that our medium supports. Instead of just video games and movies, we could add the ability to check out books and music. Additionally, we would want to completely deploy this app so that it can be accessed out of a local development environment through a provider such as Heroku and AWS. We might also want this application to update automatically by scraping different websites for new releases of different forms of media to automatically create a new listing available for rent once the media is available to the public. For example, once Nintendo releases a new video game, our application would watch their website for any new video game listings and grab all the associated information in order to create a new listing that would reflect it.

Conclusion:

The lessons learned from this project was how to structure and use a larger database with numerous tables that depend on each other. Additionally, we’ve learned how to structure relationships between a front end, back end, and database. For further possible improvements, we would want to add additional search filters so that searching by media title is not the only

way to approach it. We would also want to implement more triggers in the database to allow for the automation of specific tables instead of using business logic to do it for us. Overall, we are happy with the work that we've done and have shown how to use a database to create a sizable web project.

Division of Work:

Jose - Streamlined development environment using Docker, helped with the development of EER diagram and database schema, created API endpoints, media detail page, employee create/edit, media search, database trigger/create/querying,

Jeffrey - Front-End Sign-in/Signup, Front-End Home Page, Reusable NavigationBar/Components, Router linking, Admin create media page, Reviews(backend logic)-some api endpoint, Update profile page.

Dawson - API endpoints, FastAPI route linking, CRUD operations for database tables, Transaction View, Return functionality, database querying, DDL statements