Entertainment
Technical Design Document

1. Tech Stack

A cross-platform mobile application will be built using React Native open-source framework. React Native runs on NodeJs runtime, to build the code.
Framework & Library:

- React Native
- NodeJs
- Redux (State management)

IDE - Visual Studio Code
Version & Source Code Control - GitHub
Database - MSSQL

2. Accounts and Infrastructure

2.1 Development

The application will be developed using React Native. The repository will be created in GitHub. The development branch will be created which has the latest version of the application. The development branch will be tested and changes will be made into it. Later, the Master branch will be updated with the latest version of the development branch and will be used to deploy. For development, the source code can be cloned from GitHub and each developer will have the edit access. Developers can use VS Code as IDE to edit the source code. The application can be tested on Android and IOS physical devices. Expo-CLI should be used to run the application on the physical device. The application can also be tested on an emulator. React Native CLI should be used to run the application on a virtual device. PayPal/Stripe will be used as a payment platform that will be integrated with the application. AWS will be used to host the database server.

Technologies:

AWS(Hosting purpose)
URL: aws.amazon.com

Account: Need to be given by kevin

Data Sources, Models, Timing

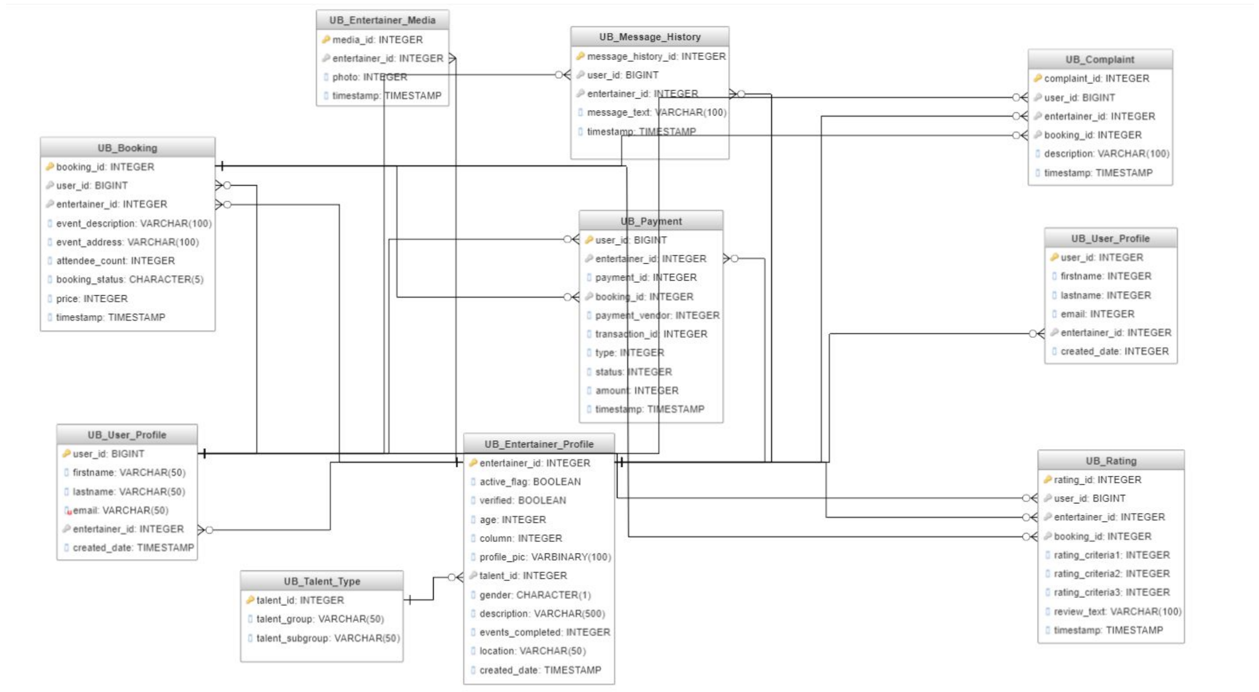
1.1 Data Sources

The following information is obtained from the client

- App logos and branding images.
- Default field list for User Profile
    - This is the information list the user provides while creating their profile.
- Default field list for Entertainer Profile
    - This is the information list the entertainer provides while creating their profile.
- Default field list for User proposal
    - The user fills out a form giving details about the event proposal
- List of talent types

1.2 Data Models and Structure

The ER Diagram is available here
https://app.genmymodel.com/editor/edit/_uXEAcFOTEeqK2M3E1LfZ7Q#



Schema definition

[UB_Settings]

The table contains static information required for the setup of the application.

Column Name	Data Type	Constraints	Description
setting_id	INT, IDENTITY	PK	
setting_name	VARCHAR(25)		
setting_value	VARCHAR(25)		

[UB_User_Profiles]

The table contains information pertaining to a user.

Column Name	Data Type	Constraints	Description
userprofile_id	INT, IDENTITY		
firstname	NVARCHAR	NOT NULL	
lastname	NVARCHAR	NOT NULL	
email	NVARCHAR	PK	
user_password	NVARCHAR		
createdAt	datetimeoffset		
updatedAt	datetimeoffset		
admin_account_flag	VARCHAR		
fb_userid	VARCHAR		
push_token	VARCHAR		
OTP	VARCHAR		
verify_flag	VARCHAR		
fb_flag	VARCHAR		
admin_account	VARCHAR		

[UB_Entertainer_Profile]

The table contains information pertaining to a user.

Column Name	Data Type	Constraints	Description
entertainer_id	INT, IDENTITY	PK	
active_flag	Boolean	Default Value = True	
verified	Boolean	Default Value = False	
age	INT		
profile_pic	BLOB		
profile_description	NVARCHAR		
events_completed	INT		
city	VARCHAR		
last_performance	VARCHAR		
type_of_entertainer	NVARCHAR		
business_id	INT		
fb_url	NVARCHAR		
insta_url	NVARCHAR		
twitter_url	NVARCHAR		
earnings	INT		
created_date	datetime		
location	NVARCHAR		
userProfile_id	INT		
typeOfBusiness	NVARCHAR		
profile_pic	NVARCHAR		
verified	VARCHAR		
payment_verified	VARCHAR		
account_id	VARCHAR		

[UB_entertainment_talent]

Column Name	Data Type	Constraints	Description
seq_numb	INT, IDENTITY	PK	
entertainer_id	INT		
talent_id	INT		
talent	NVARCHAR		
userprofile_id	INT		

[UB_Business_Profile]

Column Name	Data Type	Constraints	Description
business_id	INT, IDENTITY	PK	
active_flag	BIT		
verified	BIT		
profile_pic	NVARCHAR	NOT NULL	Eg: Dance
profile_description	NVARCHAR		
earnings	INT	NOT NULL	Eg: Ballet
created_date	DATETIME		
business_name	VARCHAR		
fb_url	NVARCHAR		
insta_url	NVARCHAR		
twitter_url	NVARCHAR		
userprofile_id	INT		

[UB_Talent_Type]

Column Name	Data Type	Constraints	Description
talent_id	INT, IDENTITY	PK	

talent_group		NOT NULL	Eg: Dance
image_url	NVARCHAR		

[UB_custom_form]

Column Name	Data Type	Constraints	Description
form_id	INT, IDENTITY		
entertainer_id	INT		
ColValue	NVARCHAR		
userprofile_id	NVARCHAR		

[UB_default_form]

Column Name	Data Type	Constraints	Description
def_form_id	INT, IDENTITY	PK	
ColValue	VARCHAR		event_name,event_location,event_time,numb_of_attendee,Monetary compensation,comments

[UB_event_proposal]

Column Name	Data Type	Constraints	Description
proposal_id	INT, IDENTITY		
event_id	INT		
entertainer_id	INT		
userprofile_id	INT		
colname	VARCHAR		
colval	VARCHAR		
col_accepted	INT		

[UB_event_booking]

Column Name	Data Type	Constraints	Description
event_id	INT, IDENTITY	PK	
userprofile_id	INT	FK, NOT NULL	
entertainer_id	INT	FK, NOT NULL	
event_description	NVARCHAR		
event_address	VARCHAR		
attendee_count	INT		
booking_status	VARCHAR		
price	INT		
event_date	DATETIME		
created_date	DATETIME		
event_name	VARCHAR		
event_privacy	BIT		
acceptedBy_entertainer	NVARCHAR		
event_otp	VARCHAR		
cancel_description	NVARCHAR		
event_verified	VARCHAR		
paid_flag	VARCHAR		
talent_id	INT		

[UB_Payment]

Column Name	Data Type	Constraints	Description
payment_id	INT, IDENTITY	PK	
event_id	INT	FK, NOT NULL	
payment_vendor	VARCHAR		

transaction_id	VARCHAR		
transaction_type	VARCHAR		Tip/initial payment/final payment
amount	NUMERIC		
created_date	DATETIME		

[UB_Rating]

Column Name	Data Type	Constraints	Description
useprofiler_id	INT		
entertainer_id	INT		
rating_id	INT, IDENTITY	PK	
booking_id	INT	FK	
rating1	INT		
rating2	INT		
rating3	INT		
rating4	INT		
review_text	VARCHAR		
created_date	DATETIME		

[UB_Message_History]

Column Name	Data Type	Constraints	Description
message_history_id	INT, IDENTITY	PK	
user_id	INT		
entertainer_id	INT		
event_id	INT		
message_text	VARCHAR		
created_date	DATETIME		

sentBy	NVARCHAR		

[UB_Entertainer_Media]

Column Name	Data Type	Constraints	Description
media_id	INT, IDENTITY	PK	
entertainer_id	INT		
media_link	NVARCHAR		
created_date	DATETIME		
TypeOfMedia	NVARCHAR		
userprofile_id	INT		

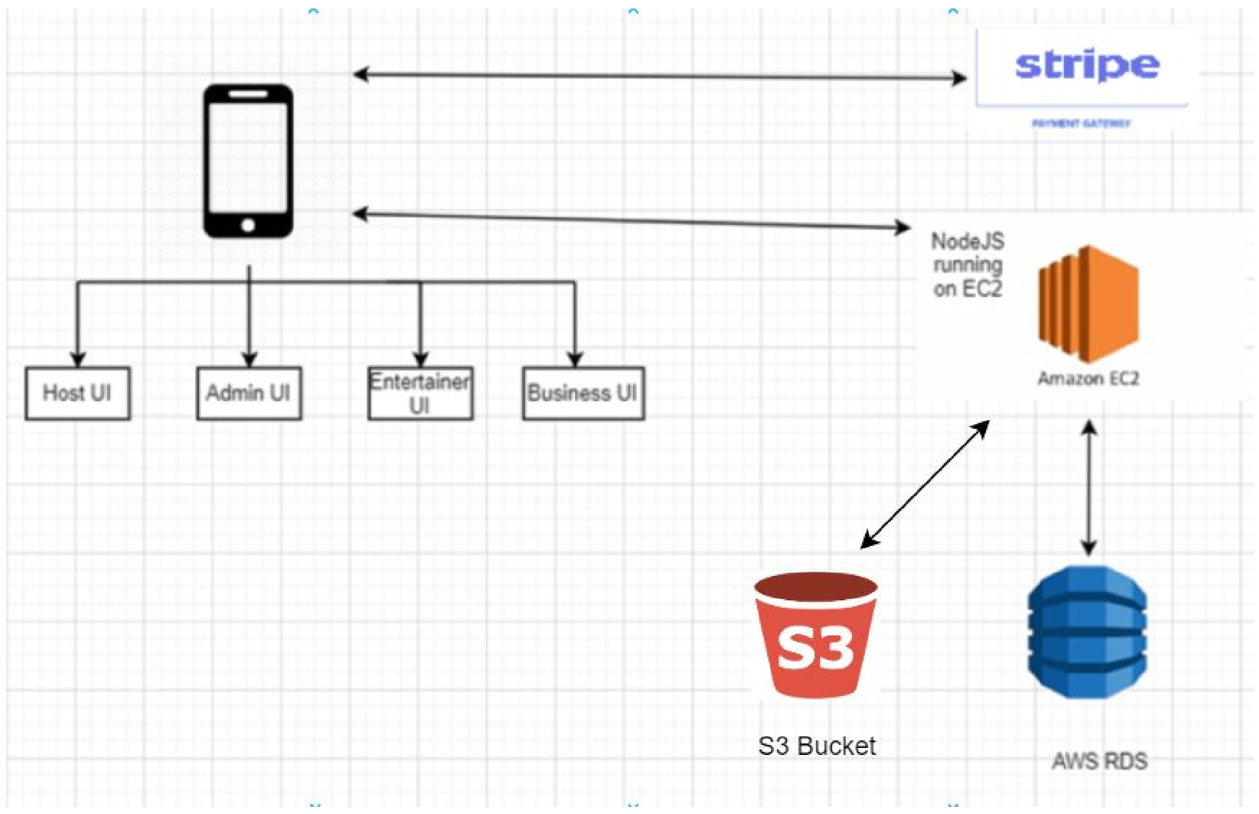
[UB_Complaint]

Column Name	Data Type	Constraints	Description
userprofile_id	INT		
entertainer_id	INT		
Complaint_id	INT, IDENTITY	PK	
booking_id	INT		
complaint_text	VARCHAR		
created_date	DATETIME		
category	VARCHAR		
response_text	NVARCHAR		

1.3 Timing

- The Personal information about the user,entertainer,admin will be on the system forever.
- Admin has the right to deactivate accounts after a period of inactivity and cancel events (to prevent potentially illegal events/activities from taking place).

System Architecture Diagram



API Layer:

1. talentCategory
   - Input - N/A
   - Output - talent_id,talent_group from UB_Talent_Type table

(This API is used to fetch a list of talent from UB_Talent_Type table).

2. loadBusinessNameList
   -Input - N/A
   -Output -business_id,business_name from UB_Business_Profile

(This API is used to load the list of businesses from UB_Business_Profile).

3. storeCustomFields
   -Input - userProfileId and field values
   -Output - N/A

(This API is used to store the additional question that the entertainer would like to add in his proposal form).

4. loadDefaultForm
    -Input - N/A
    -Output - list of questions taken from UB_default_form.
(This API is used to fetch default questions from UB_default_form).

5. uploadToS3Base64
    -Input - image in base64 format
    -Output - N/A
(This API is used to store the image in S3 bucket.)

6. createEntertainerProfile
    -Input - userProfile_id,businessName,Profile_description,Location, typeofBusiness, category, talent,fb_url,insta_url, additionalTalent, profilePic.
    -Output - N/A
(This API is used to store the form values in the create entertainer profile page in the UB_Entertainer_profile. The talent are stored in UB_entertainment_talent.)

7. createBusinessProfile
    -Input - userprofile_id, name, profile_description, fb_url,insta_url, twitter_url, profile_pic
    -Output - N/A
(This API is used to store the form values in create business profile page in UB_Business_Profile).

8. updatetalentCategory
    -Input- talent_id and talent_group
    -Output - N/A
(This API is used to update the talent value in the UB_talent_type table in DB).

9. inserttalentCategory
    -Input - talentGroup
    -Output - N/A
(This API is used to insert new talent in UB_talent_type table).
-- category.js

10. getComplaintsList
    -Input - userprofle_id
    -Output - complaint_id,userprofile_id,complaint_text, created_date, category,firstName,lastName, email, responseText

(This API is used to get the complaint raised by a user from UB_Complaint table).

11. saveResponse

-input - complaintJson
-output - N/A
(This API is used to save the response for a particular complaint in UB_complaint table).

12. getMessage
    -input - event_id
    -output - message_id,userprofile_id,entertainer_id,event_id,message_text, created_date,SentBy
(This API is used to get the list of message from UB_message_history table to display in chat)

13. storeMessage
    -input -userProfile_id,eventID, entertainerID, message, sentBy
    -output - N/A
(this API is used to store the message in the UB_message_history table).

14. loadeventForChat
    -input - userprofile_id,entertainer_id
    -output- event_name,event_id,entertainer_id,userprofile_id,firstName.

(This API is used to list the event so that the entertainer or host can chat with each other).

15. submitCOmplaint
    -input - userprofile_id, category, comment
    -output- N/A
(This API is used to store the complaints raised by users).

16. getMedia
    -input -userprofile_id
    -output - media_id, media_link, crated_date, TypeOfMedia, userprofile_id.
(This API is used to get and display the images and video).

17. uploadMedia
    -Input - imagelistJson, VideoListjson, userprofile_id
    _Output - N/A
(This API is used to save the link to image and video in UB_Entertainer_Media).

UI Layer:

The application will be built using React-Native and whenever the screen loads or any event such as button click occurs, then the API will be hit, data will be retrieved and stored in the redux store. The logos and images used for the application are saved in the source code under the assets folder.

Login/signup UI:
When the user sign ups with details and clicks on 'Sign up' button, then signup API will be called with "POST" method and data will be stored in the database. When the user logins in, then login API will be called and the user will be logged in successfully.

HomePage UI:
Whenever the user searches for an entertainer, "getSearchResults" API will he hit with the GET method. The results will be retrieved and stored in the redux store. It will be used to display the list of entertainers on the home page.

Manage Account UI:
Whenever the user clicks on manage account, "getBookingDetails" API will be hit with GET method and then the retrieved data will be used to display the past and upcoming booking details.

Profile UI:
Whenever the user/entertainer views his/her profile, "getUserProfile" API will be called with GET method and then the retrieved data will be used to display the user details. Whenever the user/entertainer edits his/her profile and clicks on submit, then the "modifyUserProfile" API will be called POST method and then the changes will be updated into the database.

Pre-requirement Form:
Whenever the host fills out the pre-requirement form and clicks on submit, the "sendRequestToEntertainer" API will be called with POST method and the changes will be stored in the database. Whenever the entertainer modifies the fields in the pre-requirement form, the changes will be saved in the database by calling the modifyPrerequirement API.

Admin UI:
Whenever Admin verifies an entertainer and confirms/deletes the account, "handleEntertainerAccount" API will be called with POST method. And the corresponding changes will be updated in the database.

Contact Us UI:
Whenever the user provides feedback or complaint, "giveFeedback" API will be called with POST method and the corresponding value will be updated in the database.