

CSCE 606: Software Engineering

A Report On CastNXT: A Web-App to Automate Talent Auditions, Placement & Replacement

Team Roles:

1. Product Owner : Vibush Karthee Shanmugam
2. Scrum Master : Ashish Singh
3. Developers : Aakash Raj Udayakumar , Ashish Singh , Varun Lokanath Pai , Vibush Karthee Shanmugam

Summary:

CastNXT is a web application to automate the management of an event (Fashion show, play and other miscellaneous events) involving talent casting. This involves 3 personas: Talent, Producer and Client. A talent (user) has the ability to register for various events and track their status of their application. The producer (event manager) has the ability to make executive decisions on the talent for an event, curate lists of talents to send to clients, remove talents from the selection pool, and negotiate talent decks with the clients. The client has the ability to suggest their top “N” preferences from the talent deck sent by the producer and negotiate based on the preferences.

Our stakeholders are FashionNXT which includes our main point of contact, Tito Chowdhury, as well as the talent, event managers and designers for the different events. Here, the need of the customer is to automate the event management process by simplifying casting of talents and negotiation with clients. This comprises shortlisting candidates for an event, curating lists of candidates for clients as well as back and forth negotiation with a client. CastNXT addresses this challenge by creating a common portal for all stakeholders that automates the event management process.

User Stories:

1. **Feature:** Complete New Application and DB Setup

As a developer,

I want to set up a new CastNXT application,

So that I can eliminate dependency on google authentication (login), Google Forms (data collection), and Google Spreadsheets (data storage).

Points: 3

Duration: Iteration 0 to 1

Implementation Status: Completed

Justification:

The CASTNXT application is a legacy application that we inherited from a previous team. The first step to begin work on this application was to complete the setup of the current application and take stock of current progress. To do this, we needed to complete the application and database setup to run the application on our machines.

2. **Feature:** Develop Sign-Up Feature for the 3 Personas

As a user,

I want to be able to sign-up for the CastNXT application,

So that I can access content exclusive to my role.

Points: 5

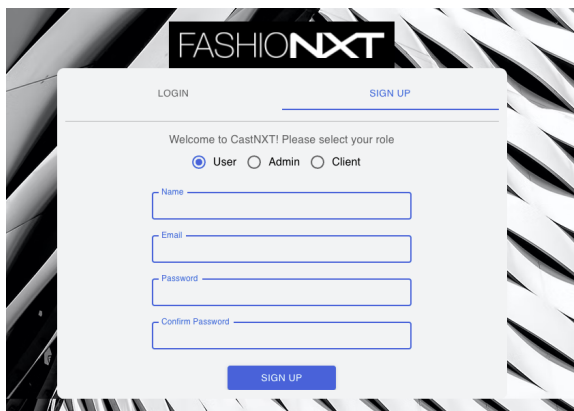
Duration: Iteration 0 to 1

Implementation Status: Completed

Justification:

As a team, we made a decision to move away from the existing state of the application, which heavily relied on Google's suite of tools (Authentication, Forms and Sheets) to run. We decided to develop our own in-house application and as a result, we needed to design our own sign up feature that connected to our local database.

Screenshot:

The screenshot shows a web application interface for 'FASHIONNXT'. At the top, there's a header with the 'FASHIONNXT' logo. Below the header, there's a navigation bar with 'LOGIN' and 'SIGN UP' links. The 'SIGN UP' link is highlighted. The main content area displays a sign-up form. The form starts with a welcome message: 'Welcome to CastNXT! Please select your role'. Below this, there are three radio buttons for role selection: 'User' (selected), 'Admin', and 'Client'. The form then has four input fields: 'Name', 'Email', 'Password', and 'Confirm Password'. At the bottom of the form, there is a blue 'SIGN UP' button. The background of the page features a stylized, abstract pattern.

3. **Feature:** Develop Log-In Feature for CastNXT

As a user,

I want to be able to login to my dashboard,

So that I can access my previously created account.

Points: 5

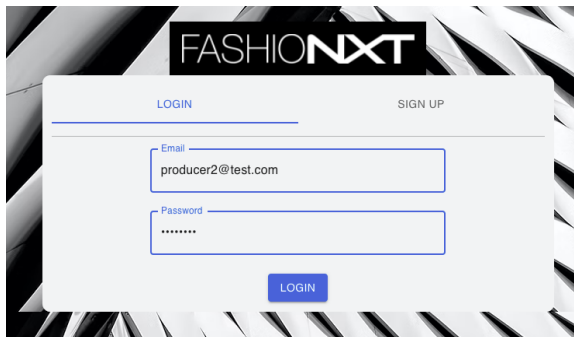
Duration: Iteration 0 to 1

Implementation Status: Completed

Justification:

The previous version of CASTNXT made use of Google Authentication in order to login to the application. The entire flow of the application involved the use of Google. This did not leave much room for user customization to various features of the application and hence, we took the executive decision to move away from Google integration and make use of our own in-house application built from the ground up. We removed the old login feature via Google and developed our own Login functionality.

Screenshot:



4. **Feature:** Integrate New Authentication with Legacy Dashboard

As a developer,

I want to connect the new authentication with the legacy persona-specific dashboard,

So that I can transition faster to the new app development without relying on legacy authentication.

Points: 0

Duration: Iteration 0 to 1

Implementation Status: Completed

Justification:

We wanted to preserve the existing personas in the application while also working on developing our revamped application. As a result, our new authentication system still needed to work with the user profiles from the previous code to preserve the functionality of each user. This user story was repurposed

into 'API Integration between Login/Sign-Up screens and backend' since we designed and revamped the application.

5. **Chore:** Learn the existing integration of Google services. To this end, we need to set up a meeting with the previous owners of the project (SE Batch Fall 2021).

Points: 0

Duration: Iteration 0 to 1

Implementation Status: Not Applicable

Justification:

When we inherited CASTNXT from the previous team, the application made use of Google Authentication to sign in, Google Forms to create forms for an event and Google Sheets to view and manage submissions for an event. Very little was actually being handled in the web application and most of the scope was pushed outside of the application via the Google Suite. As a chore, we wanted to understand why this decision was made and contacted the previous team to gain more insights. We also spoke to Tito, our stakeholder and this gave us a good understanding of where the application stood and all the features that were still pending.

6. **Feature:** Review non-relational DB design

Points: 3

Duration: Iteration 2

Implementation Status: Completed

Justification:

Created a mongoDB schema for the functionalities in the application. Reviewed the proposed design with the stakeholder and obtained approval.

7. **Feature:** Heroku Container Deployment

Points: 3

Duration: Iteration 2

Implementation Status: Completed

Justification:

The CastNXT application runs on Heroku using Docker. The application has been rebased to use MongoDB.

8. **Feature:** MongoDB Integration

Points: 2

Duration: Iteration 2

Implementation Status: Completed

Justification:

The application has been rebased to make use of MongoDB. Removed all traces of ActiveRecord to make use of “mongoid”. Addressed the chore involved under the “Legacy Project Story Refactoring” section. This change was made since most of the application involved the use of dynamic data that can change from event to event. This would not be easy to map into a relational database.

9. **Feature:** Support dynamic form creation in producer persona

Points: 5

Duration: Iteration 2

Implementation Status: Completed

Justification:

Basic front-end was completed as part of this story. We decided on the final schema structure for form creation under the producer persona. However, the API for persisting the form skeleton is currently under development. A story was added to address this and the integration with the front end. Changed the points from 8 to 5.

Screenshot:

Step 2

CHOOSE EXISTING FORM

CREATE NEW FORM

New Input 1

Object Name ⓘ

Display Name ⓘ

Name

Name

Description ⓘ

Input Type ⓘ

Description

Short Answer

Default value

Default

Required

Form preview:

New Input 1

Submit

CREATE EVENT

10. **Feature:** Support event form viewing and submission in talent persona

Points: 5

Duration: Iteration 2

Implementation Status: Completed

Justification:

We decided to save the form data as a JSON string in a field under the Submission document in our Mongo database. API integration was taken up in Iteration 3. Changed the points from 8 to 5.

Screenshot:

The screenshot shows a web form titled "Event Registration". At the top, there is a blue button labeled "BACK TO HOMEPAGE". Below this is a rounded rectangular box containing the form. Inside the box, the title "Test event with image" is displayed. Underneath the title, the text "Test event with image" and "potrait" are shown. There is a file upload section with a "Choose File" button and the text "No file chosen". Below this is a text input field labeled "name". At the bottom of the box is a blue "Submit" button.

11. Feature: API Integration between Login/Sign-Up screens and backend**Points:** 5**Duration:** Iteration 2**Implementation Status:** Completed**Justification:**

The Login/Sign-Up screens now redirect authorized users to their appropriate homepages. User session is also made use of to allow multiple windows.

12. Feature: List and view events and details for all 3 Personas**Points:** 8**Duration:** Iteration 3**Implementation Status:** Completed**Justification:**

Talent's event list has been split into two tabs: Available (Accepting events without submission) and Submitted (Submissions to still accepting events can be editable. Submission status is displayed as well).

Producer's event list consists of all events owned by them. Status of each event is displayed as well. Producer also has an option to create an event.

Client's event list consists of all events assigned to them.

Screenshots:

As a producer:

FashionNXT Events

CREATE NEW EVENT

Event	Status
Paris Fashion Week	ACCEPTING
Event 1	ACCEPTING
Broadway - Harry Potter and the Cursed Child	FINALIZED

As a talent:

FashionNXT Events

EVENTS

SUBMISSIONS

Events
Paris Fashion Week

As a client:

FashionNXT Events

Event	Status
Paris Fashion Week	ACCEPTING
Event 1	ACCEPTING
Broadway - Harry Potter and the Cursed Child	FINALIZED

13. **Feature:** API Integration for form submission For User Persona

Points: 8

Duration: Iteration 3

Implementation Status: Completed

Justification:

A form for the event is generated using saved templates. Talent can fill out fields to register for an event. Talent can edit a submission to an event. Fields of the form are pre populated in case of an edit.

14. Feature: View saved forms when creating a new event for Producer Persona**Points:** 5**Duration:** Iteration 3**Implementation Status:** Completed**Justification:**

Producer can choose a form template or create a new one after entering the event title and description. Form ids are available via a dropdown to choose a template. Preview is available before choosing.

Screenshot:

Step 2

CHOOSE EXISTING FORM

CREATE NEW FORM

Form number

Form 6277eefda9b87f000528f818

LOAD THIS FORM

Form preview:

Name

Date of Birth

mm/dd/yyyy

Portfolio

Submit

CREATE EVENT

15. Feature: API Integration for Dynamic Form Creation**Points:** 8**Duration:** Iteration 3**Implementation Status:** Completed**Justification:**

A producer can assign clients to an event via a multi-select dropdown. Producer can choose a template or create a new one. This architecture allows a single form template to be shared by multiple events. The custom form-builder library generates a JSON schema. Once the event is created using a pre-existing form available to the producer or using a new template, the event will be visible as 'ACCEPTING' submissions in the user persona.

16. Feature: Curate master deck from initial list of talent submissions in admin persona

Points: 5

Duration: Iteration 4

Implementation Status: Completed

Justification:

As an event manager, you have the ability to select/reject a talent from the initial pool of submissions. The accepted candidates are filtered into a stack known as the master stack/deck for an event.

Screenshot:

Broadway - Harry Potter and the Cursed Child

Auditions open for a two-part play written by Jack Thorne based on an original story by J. K. Rowling.

HOME SUBMITTED DOCS SELECTED DOCS CLIENT DECKS FINALIZED DECKS SUMMARY BACK TO FORMS

Use this page to create a master slide deck for this event.

Email Address*
lucy27@test.com


Name*
Lucy Hammond

Age
27

Portfolio
Choose File No file chosen

• IMG_7885-800x1200.jpg (image/jpeg, 384747 bytes)

Uploaded image



ADD

Rows per page: 1 1-1 of 4

CURATE STACK

17. **Feature:** Add a button in event for producer persona to begin talent submissions review and stop accepting new submissions

Points: 3

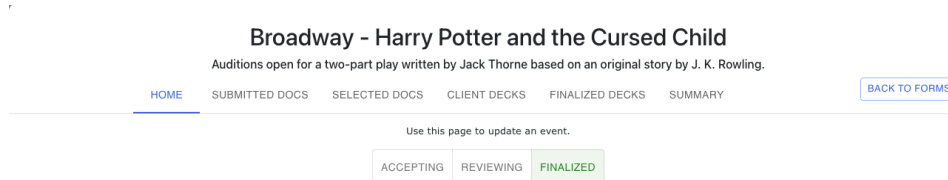
Duration: Iteration 4

Implementation Status: Completed

Justification:

As a producer, you have a button on the home screen of an event that toggles the state of an event between “Accepting”, “Reviewing” and “Finalized”. While the event state is “Accepting”, candidates can still register for the event. When the event status is “Reviewing”, talents can no longer make submissions to the event and negotiation begins between the producer and clients. When the event status is “Finalized”, negotiation between the producer and clients is complete and clients can no longer post comments or update their preferences for an event. The task to prevent talent from editing/submitting their event form when the event is in ‘Reviewing’ state was incomplete and we have opened a new story in Iteration 5 for the same (more details in ‘Unfinished User Stories’ section).

Screenshot:



18. **Feature:** Assign talents to client decks for negotiation in admin persona

Points: 8

Duration: Iteration 4

Implementation Status: Completed

Justification:

The Producer can assign clients to individual slides in the master deck for an event. The Producer can then view the curated decks for each client in the “Client Decks” screen. He/she also has the ability to edit these decks by adding or removing slides from a client deck.

Screenshot:

Broadway - Harry Potter and the Cursed Child

Auditions open for a two-part play written by Jack Thorne based on an original story by J. K. Rowling.

[HOME](#)
[SUBMITTED DOCS](#)
[SELECTED DOCS](#)
[CLIENT DECKS](#)
[FINALIZED DECKS](#)
[SUMMARY](#)
[BACK TO POP](#)

Use this page to create client-specific slide decks

Name*

Tim Horton

Age

30


Portfolio

Choose File

No file chosen

- R.jpg (image/jpeg, 1257809 bytes)

Uploaded image



Clients:

Kate Johnson

×

^

Search

☐ Select All

☐ Sam

☒ Kate Johnson

UPDATE DECKS

19.Feature: Media Field Integration in Event Registration (Talent) and Creation (Producer)

Points: 8

Duration: Iteration 4

Implementation Status: Completed

Justification:

A talent must have the ability to upload images to his/her portfolio when registering for an event. This portfolio typically includes more than one image depending on the requirements of the event. The producer for the event must then have the ability to see the portfolio uploaded by each talent so that he/she

can begin the process of shortlisting candidates for the event and carry out negotiation with the clients.

20. **Feature:** Update talent application status when selected/rejected/under review in talent persona

Points: 2

Duration: Iteration 5 (Wrap-up)

Implementation Status: Completed

Justification:

A talent must have the ability to view their event form status once the event selections are finalized based on the initial curation from the admin and the negotiation between the admin and the client. This can be seen in their submitted forms tab.

Screenshot:

FashionXT Events	
EVENTS	SUBMISSIONS
Event	Status
Event 1	SUBMITTED

21. **Feature:** Summary of final client decks after negotiation

Points: 8

Duration: Iteration 5 (Wrap-up)

Implementation Status: Completed

Justification:

An admin/event manager should be able to see the final selected talents by individual clients and the overall unique talent list after negotiations conclude with the various clients for an event. Two separate views have been created to achieve this as a talent can belong to multiple client decks. Hence a unique list of selected talents is needed as well. This will help the admin better track the selected talents for payout/assigning roles/shortlisting for other events etc., all of which are extensive features which can be picked up by a future team.

Screenshots:

Broadway - Harry Potter and the Cursed Child

Auditions open for a two-part play written by Jack Thorne based on an original story by J. K. Rowling.

[HOME](#)
[SUBMITTED DOCS](#)
[SELECTED DOCS](#)
[CLIENT DECKS](#)
[FINALIZED DECKS](#)
[SUMMARY](#)

[BACK TO FORMS](#)

Finalized Decks

Select a client below to view their finalized deck

Sam

Name	Email Ad...	Name	Age	Portfolio
Johanna Rodriguez	johanna@test.com	Johanna Rodriguez	24	data:image/jpeg;n...

1-1 of 1 < >

Broadway - Harry Potter and the Cursed Child

Auditions open for a two-part play written by Jack Thorne based on an original story by J. K. Rowling.

[HOME](#)
[SUBMITTED DOCS](#)
[SELECTED DOCS](#)
[CLIENT DECKS](#)
[FINALIZED DECKS](#)
[SUMMARY](#)

[BACK TO FORMS](#)

Event Summary

Name	Clients assigned	Email Ad...	Name	Age	Portfolio
Tim	Kate Johnson	tim@test.com	Tim Horton	30	data:image/jpeg;n...
Johanna Rodriguez	Sam, Kate Johnson	johanna@test.com	Johanna Rodriguez	24	data:image/jpeg;n...

1-2 of 2 < >

22. Feature: Prevent ability to submit if event is under review

Points: 8

Duration: Iteration 5 (Wrap-up)

Implementation Status: Completed

Justification:

An admin/event manager should be able to freely control the state of the event creation, selection and finalization process without interruptions from the talent and client personas. To this end, we need to control when the user should be stopped from submitting/editing event applications and when the clients should be prevented from further altering their feedback with respect to a particular talent.

Strategy to improve prior code

The current technology stack for the application makes use of HTML and JavaScript for the UI and Rails for the backend. The major part of the functionality in this application is handled through the use of google forms that then automatically populate google spreadsheets. The communication between stakeholders takes place through these google spreadsheets.

In our implementation, we moved away from this dependency on google services by doing the following:

- Eliminating the use of google for authentication by using our own DB to manage users and their roles
- Eliminating the use of google forms by creating our own UI to create forms for different events.
- Moving away from the google suite to manage talent for events by shifting to a Database(decided as MongoDB to handle continual change and provide flexibility in Iteration 2) to maintain all information.

The above changes enable the entire lifecycle of the talent casting, selecting and negotiation process to remain on a single CastNXT application dashboard, thus, preventing context switching and relying on Google services for storage, which comes with its own set of limitations.

In addition to the above improvements, we also made the following general changes:

- Revamped the entire UI to make it more user friendly and intuitively capture the 3 personas - the talent submitting their forms, the event manager handling the event flow and the clients giving their talent preferences by negotiating with the event manager.
- Completed the unimplemented functionality from the previous semester
- Developed our own database schema to manage user and event related data.

Team Roles

1. Product Owner : Vibush Karthee Shanmugam
2. Scrum Master : Ashish Singh
3. Developers : Aakash Raj Udayakumar , Ashish Singh , Varun Lokanath Pai , Vibush Karthee Shanmugam

The team roles remained consistent and the nature of work, dynamic, through the course of the semester. We worked as a closely knit group with multiple people usually working on the same story to deliver it faster. Before the commencement of the iterations, we went over the tasks that need to be completed over the course of the iteration and helped segregate the work to prevent dependency conflicts.

Scrum Summaries

Iteration 1:

The goal of this iteration was to gain a complete understanding of the legacy project. A major chunk of the sprint was spent in refactoring the code to remove dependency on Google authentication which was failing and establish personas to segregate the work based on talents, event managers and producers. We developed and demoed the UI for login and signup features and established a connection between the new authentication workflow and the old legacy homepages.

Iteration 2:

We noticed several feature bugs in the legacy application and the UI seemed unintuitive to us. We as a team decided that instead of spending time and effort to correct previous integration as well as fix UI, we'll rework the features of the application from scratch in an easy-to-understand intuitive manner.

We discussed new and improved features to be added from the perspective of 3 types of users of the application: talent, event manager and client. Based on the requirements, we decided to go with MongoDB (non-relational DB) as our backend of choice to continuously meet the requirements of the dynamic nature of changing event forms. We designed the non-relational DB schema and UI mockups of the application and got them reviewed and approved by the stakeholder. We also developed the front-end for the event form submission from the talent perspective, event form creation from the event manager perspective and login/signup validations for the 3 personas, each pointing to a different home page view.

Iteration 3:

Based on the UI developed in the previous, we integrated the event creation and submission with our MongoDB backend. We also added the ability to preview previously created form skeletons so that they could be reused when creating new events in the future. Over this iteration, we made several changes to our schema based on requirements from the client. Using MongoDB made it extremely efficient to build on top of existing data.

Iteration 4:

We added the ability to upload portfolio images when creating and submitting event forms (from event manager and talent perspective respectively). We also added the ability to curate decks to form the initial master deck for a particular event - this meant

the first phase of rejecting talent submissions from the event by the event manager. We also provided a feature in the event manager persona to add talents to a particular client deck based on which negotiation can commence between the event manager and the client. Based on this feature, we also demoed the client home page which showed the list of events in which the client has received updated decks from the admin to review and provide feedback. We also added logic to control the entire process flow between user -> event manager -> client using event flags like “ACCEPTING”, “REVIEWING” and “FINALIZED” to indicate the current status of the event. In the ‘ACCEPTING’ phase, users can submit and edit form submissions. In the ‘REVIEWING’ phase, submissions from talents is disabled and they can only view their submission status as ‘UNDER REVIEW’. Once the event is in review mode, the admin and the client can begin negotiation on the assigned deck. In ‘FINALIZED’ mode, no further changes to the event are allowed and the client cannot provide any more feedback. We also started working on the major negotiation feature over the course of this iteration which we completed in Iteration 5 (wrap-up phase).

Iteration 5 (Wrap-up):

We completed the negotiation feature between the admin and the client using a preference(priority) list provided by the client; the UI provided the ability to reorder the talent deck assigned by the event manager. This list would be updated in the admin/event manager persona to reflect the feedback, based on which executive decision can be taken by the event manager whether to finalize a talent for a particular client or not. We also added event and client summaries features to reflect the finalized talents by client and the overall unique talents list accepted for the event. We also reflected the talent application status in the talent persona whether they have been accepted or rejected. This wrapped up reworking the entire talent, event manager and client flow from the creation of an event to its conclusion.

Customer Meetings

Meeting 1: Problem Description and Goals

Date: 25th Feb '22

Time: 5-6.40 PM

Minutes of the meeting:

1. Discussed challenge in setting up google authenticator for the project from previous semester's work.

2. Discussed and understood roles of user (talent), admin (event manager) and client (designer) and their respective dashboards.
3. Discussion on whether to drop Google Forms: Right now, the web app is just redirecting info from the app to the google form/spreadsheet for data editing/manipulation. In case of no redirection, it uses a model of the same data as in the Google Form/Spreadsheet to use on the WebApp.
4. Reviewed sample completed google spreadsheet decks after the entire workflow.

Action Items:

1. Set up a call with the previous team to discuss the challenges faced in google authentication and review missing information for current project set-up.
2. Need to decide whether to continue dependency on Google forms/spreadsheets for data persistence/editing or switch focus towards building a DB to be used within the CastNXT App.

Meeting 2: Legacy App Bugs and New App Future Outlook

Date: 4th Mar '22

Time: 5-6.15 PM

Minutes of the meeting:

1. Had a demonstration of the previous teams' app and highlighted the various bugs on each screen including authentication (invalid access allowed) bugs.
2. Understood a simpler use case for the web app - negotiation for event resources (could be talent, set props, make up artists etc) between the client and casting producer/manager.
3. Discussed new app features to be added (NOTE: This is currently on the backburner until the legacy app bugs are fixed and new authentication model is developed):
 - Communication thread on a model in a deck sent to client on the app rather than using email
 - Notification to client/producer via email when there's a new comment/question on a model
 - Make decks uneditable (by producer) when close to event
4. Outlook on the first phase of the project - goals to be achieved

Action Items:

1. Discuss a plan to build the wireframe with new authentication system in place

2. Ramp up to existing state of the project (needs overhaul as it is riddled with bugs) - build out stories accordingly

Meeting 3: Login/Registration and Initial Wireframe

Date: 9th Mar '22

Time: 5-5.35 PM

Minutes of the meeting:

1. Had a demonstration of the user login/registration feature for the 3 personas - highlighted redirection issue, currently being worked on.
2. Highlighted SQL and NoSQL DB setup requirements for the web app, currently being worked on.
3. Plan to build the form creation(admin) with support for JPEG/PDF, submission (talent) and persistence as part of the next iteration using either Google forms (interceptor to capture form data in JSON) or use a 3rd party form builder library.
4. Next meeting after spring break - 23rd March 2022.

Action Items:

1. Include a design review story as part of the next iteration.
2. Resolve redirection errors in wireframe, complete DB setup.

Meeting 4: Non-relational DB schema review

Date: 23rd Mar '22

Time: 5-6.30 PM

Minutes of the meeting:

1. Started updates on project tracker - https://docs.google.com/spreadsheets/d/1YJLCJufmIHjFyzbBvxfoLzX_StdclPJxqv6fDJ-PubU/edit#gid=1650533702.
2. Changes to schema based on questions/comments:
3. "specs" and "media" and other variable fields will all be present as part of another field object called "form_data". This is needed as the form itself is variable for each event, hence may not contain the same data.
4. "producers" array to be CHANGED to "producer" field with single value - for simplicity. Array conversion can be implemented later.
5. "client_deck_rel" REMOVED as each client will be assigned only a single deck.
6. state of form changed to ACCEPTING/REVIEWING/FINALIZED.

7. "final_decks" REMOVED as "deck_clients" will maintain the final client a talent is assigned to, hence duplication won't be necessary.
8. provisioning needed for adding comments and chat history on "TOP" slide which is not relevant to any individual talent but the deck as a whole - NOT A PRIORITY.

Action Items:

1. Update DB schema as per comments.
2. Push DB schema changes to code base.
3. Create a "form_skeletons" collection to store skeletal form structures which the producer creates. When creating a new event, he/she should be able to use a preexisting skeletal structure available to him/her.

Meeting 5: Event Creation Demo

Date: 30th Mar '22

Time: 5-6 PM

Minutes of the meeting:

1. Demonstrated UI for event creation which involved creation of field names and types grouped by sections.
2. Demonstrated talent/user home page view to see all events.
3. Faced issues in backend integration with how the event form skeleton (in admin persona) should be getting saved.

Action Items:

1. Form model will now save the schema as a JSON string to facilitate simpler fetch and push. This is done so that the JSON schema to UI converter can be used for form preview in admin persona as well as loading event registration in user persona. The schema JSON will be used to build the UI rather than saving individual key value pairs for each field (and it's type, eg - {"fieldTitle": "Name", "fieldType": "textfield"}) and using those to build UI.

Meeting 6: User View All New Events and Submissions Demo

Date: 6th April '22

Time: 5-6.15 PM

Minutes of the meeting:

1. Demonstrated talent/user home page view to see all events and view all submissions along with status with mock data.

2. Demonstrated form creation with new media field type option to upload images.

Action Items:

1. Update the new media field type to show an image preview when an image/pdf is uploaded when creating a new form under Admin/Producer persona.
2. Having Submission and Slide models separate to represent Talent and Admin view of the same talent portfolio led to complicated integration on the backend. Decided to merge the two collections together under Slide. The user submission will now go as "data" under the Slide collection. The data will be used as a JSON string similar to how the form skeleton is saved to ease form view for editing for user persona.

Meeting 7: Master Deck Curation Demo

Date: 20th April '22

Time: 5-6 PM

Minutes of the meeting:

1. Demonstrated image upload feature for an event
2. Demonstrated the creation of a master deck for an event.
3. Demonstrated talent/user home page view to see all events.

Action Items:

1. Creation of client decks. Once a producer creates a master deck for an event, he/she should have the ability to create individual client decks.
2. Add support for viewing slide decks for each client. Producer should have a tab where he/she can view the curated slide decks for each client.

Meeting 8: Overall Slide curation and individual persona features Demo

Date: 27th April '22

Time: 5-6.15 PM

Minutes of the meeting:

1. Demonstrated slide curation flow - creation of master deck and individual client decks and viewing individual client decks.
2. Demonstrated functionalities for all 3 personas and walked through the actions of each persona.

Action Items:

1. Few visual changes to some of the wording for tabs and buttons in the UI. Tito has provided us with feedback on some changes that he would like to see.
2. Change client and admin sign up. By default, only users(talents) should be able to sign up via email address and password. The admin and client roles will have to be approved and created by a default admin now.
3. Admin should support the ability to add new client credentials. Option to add clients in the event creation screen will be removed. Instead, admin will now add clients from the entire list directly after the initial talent shortlist is created.
4. Summary page for an event. Need to create an overall summary page for an event, both from the client perspective and the admin perspective. From the admin perspective, this should display the selected candidates and the clients associated with the event. From the client perspective, this should allow a client to order his preferences of talent for an event.
5. Support option to edit an existing form for an event.
6. Fix bug in client view - the client persona currently displays events that they may not be a part of.

BDD/TDD Process

Process:

We started working on top of the unit tests written by the previous team. The existing unit tests were written using the default rails unit testing framework called "Minitest". Due to Minitest's incompatibility with MongoDB, we switched to RSpec. We configured RSpec to work with a mocked test instance of MongoDB, where the test database will be cleared after each test suite to give a clean start. We followed TDD by creating tests for every controller which would call a series of endpoints to setup the DB. The test would expect the controller to create/get/update specific resources with appropriate authentication and authorization. We would write the controller to satisfy the above test and also make sure it does not break any other tests. This way both the functionality of the controller as well as it's compatibility with other endpoints was tested.

Challenges:

1. We started the project with SQLite backend which was tested with MiniTest framework. Since we realized the need for MongoDB and made a switch in the backend, Minitest framework became incompatible with MongoDB. Thus we switched to RSpec as the primary testing framework. This switch meant we had to setup RSpec and rewrite some of the important unit tests from Minitest.

2. Since we needed a clean state of database for every test suite, we had to clear the database after every run of the test suite. In order to do it efficiently, we used a library called the “database_cleaner” and configured it to clean the database after every test suite. Choosing the correct library was a challenge since it had to be compatible with both MongoDB as well as RSpec.

Benefits:

1. Since some of our endpoints were available for multiple user personas, we had to ensure that the controller responded according to each user’s authorization level. Having unit tests which checked based on authorization of each persona, made it easy to write controllers that were aware of authorization levels.
2. Having unit tests also served as a guide for frontend developers during integration. Referring to unit tests gave frontend developers an idea on the parameters that were needed to be passed to each endpoint.

Configuration management

MongoDB (Backend):

The legacy project was using SQLite as the backend and thus the default ActiveRecord was used as the Object Relational Mapper (ORM). In order to use MongoDB as the backend, ‘mongoid’ ruby gem was utilized which replaces the default ActiveRecord as the Object Document Mapper (ODM). For this to work, we needed to replace all traces of ActiveRecord in the legacy code.

This was accomplished by the following:

1. Commenting references to active_record in config directory

```
config.active_record.dump_schema_after_migration = false
config.active_storage.service = :local
```

2. Removal of active_storage npm package

```
yarn remove @rails/activestorage
```

3. Updating application.rb to pull specific frameworks by changing require "rails/all" to:

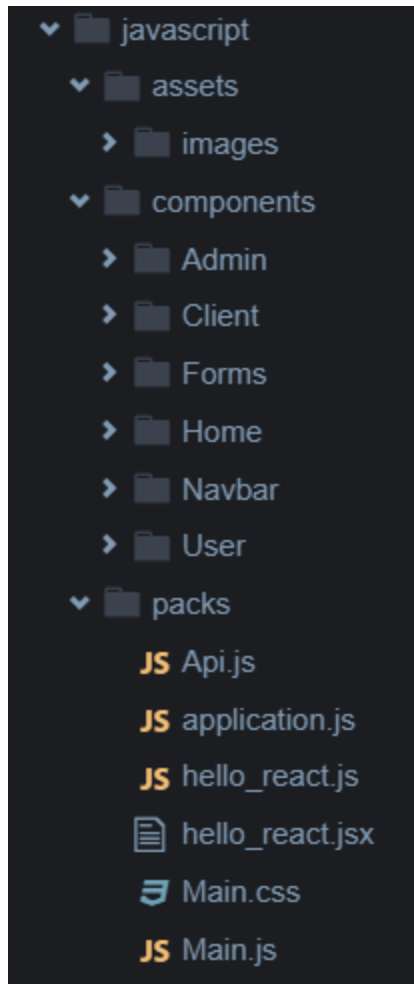
```
# Pick the frameworks you want:
require "active_model/railtie"
require "active_job/railtie"
# require "active_record/railtie"
# require "active_storage/engine"
require "action_controller/railtie"
require "action_mailer/railtie"
require "action_mailbox/engine"
require "action_text/engine"
require "action_view/railtie"
require "action_cable/engine"
```

4. Removing this line from application.js webpack entry point file

```
require("@rails/activestorage").start();
```

React (Front-end):

We replaced the typical views in Ruby on Rails in favor of React.js for the front-end of the application. All packages for the front end were managed using the package.json file. The file structure for the front end code is shown in the screenshot below:



In order to use React, we had to change the setup of the application by installing webpacker to compile React for the frontend. Rails hits the `hello_react.js` file to render react. `hello_react.js` in turn renders `Main.js`, which is a file defined by us and points to all the other frontend code files.

The actual React files are stored in the components folder and each sub-folder in components corresponds to a major functionality in the application. Each subfolder contains the corresponding javascript files that implement the various functionality of the application.

In order to link React with the backend, we learnt that `react-router-dom`, which is typically used for navigation in React, doesn't allow the application to hit the backend. Hence, we navigate through react by setting the URL using `window.location.href`.

Additionally, to link the frontend with the backend, for each controller in the backend, a corresponding <controller_name>.html.erb file will have to be created in /views/layouts with the following text pasted in:

```
<!DOCTYPE html>
<html>
  <head>
    <title>ReactRails</title>
    <%= csrf_meta_tags %>
    <%= csp_meta_tag %>
    <%= stylesheet_link_tag 'application', media: 'all',
'data-turbolinks-track': 'reload' %>
    <%= javascript_pack_tag 'application', 'data-turbolinks-track':
'reload' %>
    <%= javascript_pack_tag 'hello_react' %>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

Production Issues

The deployment is currently running on the Development Environment. There are some configuration issue with our new routes that don't allow Production Environment access.

The deployment is container-based. To deploy on vanilla Heroku we require the app to be in the root of the repository or for AWS to support git-subtree. Since we have neither, the deployment becomes inherently more difficult.

We also needed to run pre deployment scripts such as “**rails webpacker:install**” and “**rails webpacker:compile**”. The changes from these commands were not persistent in the vanilla Heroku deployment. Hence, we decided to switch to Heroku Container deployment to persist our pre deployment scripts.

The deployment requires a connection to MongoDB Atlas. Heroku Container deployments do not allow MongoDB within containers (Due to this, we also do not have an idea about the persistence of data). To resolve this we made use of a Free MongoDB Atlas cluster for our database.

AWS and Github Issues

AWS Free tier offers 10GB Storage and 1GB RAM. There are some issues that arise due to this:

- Storage: This is not enough for both deployments. You will need to update your storage to 25GB before proceeding. (AWS has a Free Storage Limit of 30GB)
- RAM: This is not enough for the Heroku Container deployment. You will run into **insufficient heap storage** when containerizing the application. You will need to upgrade to **t3.small** (Create this instance only for deployment. It is **NOT FREE**). 1GB RAM from the Free Tier will be sufficient for the Local AWS deployment.

Deployment Guide

Local AWS

Clone the Repository

```
git clone https://github.com/aakashraj96/CASTNXT.git
cd CASTNXT/web/CASTNXT
```

Install MongoDB 5.0

```
sudo tee /etc/yum.repos.d/mongodb-org-5.0.repo<<EOL
[mongodb-org-5.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/amazon/2/mongodb-org/5.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-5.0.asc
EOL
```

```
sudo yum install -y mongodb-org
sudo systemctl start mongod
```

Install Ruby and Node Modules

```
rvm install "ruby-2.6.6"
bundle install
```

```
npm install -g npm@8.5.4
nvm install 16.13.0
npm install -g yarn
bundle exec rails webpacker:install
```

Deploy the Application

```
rails s -p $PORT -b $IP
```

Preview the running application using AWS IDE.

Heroku Container

Clone the Repository

```
git clone https://github.com/aakashraj96/CASTNXT.git
cd CASTNXT/web/CASTNXT
```

Create a Free MongoDB Atlas Cluster

[CLUSTERS](#) > CREATE A SHARED CLUSTER

Create a Shared Cluster

PREVIEW Serverless	Dedicated	Shared
---------------------------	-----------	--------

FREE Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

[Back](#) [Create Cluster](#)

Configure the Free MongoDB Atlas Cluster under **Security Quickstart**

How would you like to authenticate your connection?

Your first user will have permission to read and write any data in your project.

Username and Password

Certificate

Create a database user using a username and password. Users will be given the *read and write to any database privilege* by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password. You can manage existing users via the [Database Access Page](#).

Username

username

Password

password

Autogenerate Secure Password

Copy

Cancel

Update Password

Username	Authentication Type
username	Password

EDIT

Where would you like to connect from?

Enable access for any network(s) that need to read and write data to your cluster.

My Local Environment

Use this to add network IP addresses to the IP Access List. This can be modified at any time.

Cloud Environment

Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.

Set your network security with any of the following options

Only an IP address you add to your Access List will be able to connect to your project's clusters. You can manage existing IP entries via the [Network Access Page](#).

IP Address	Description
Enter IP Address	Enter description

Add Entry

Add My Current IP Address

IP Access List	Description
0.0.0.0/0	All

REMOVE

VPC Peering

Peer your VPC with your Atlas cluster's VPC to ensure that traffic does not traverse the public internet. Requires an M10 cluster or higher.

Configure in New Tab

Private Endpoint

Use your Private Endpoint to create a one-way connection from your VPC to your MongoDB Atlas VPC, ensuring Atlas cannot initiate connections back to your network. Requires an M10 cluster or higher.

Configure in New Tab

Connect to Heroku and Create a Deployment

```
heroku login -i
heroku container:login
heroku create -a <app_name>
```

Push code into the Deployment

```
heroku container:push web -a <app_name>
heroku release web -a <app_name>
```

Get MongoDB Atlas Connection URL

Connect to Cluster1

Setup connection security Choose a connection method Connect

1 Select your driver and version

DRIVER	VERSION
Ruby	2.5 or later

2 Add your connection string into your application code

Include full driver code example

```
mongodb+srv://username:password@cluster1.8akze.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
```

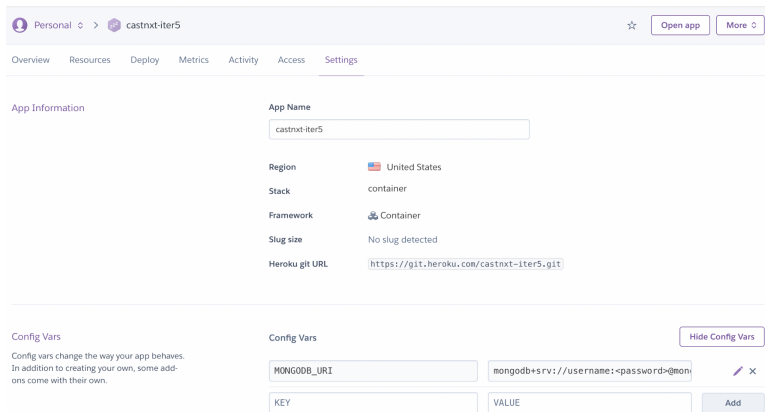
Replace `password` with the password for the `username` user. Replace `myFirstDatabase` with the name of the database that connections will use by default. Ensure any option params are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back

Close

Connect Deployment to MongoDB Atlas on Heroku.com



Open App to view Deployment

Gems/Tools Used

1. **SimpleCov:** We used SimpleCov to track the code coverage of our tests. The convenience that this gem provided was that the integration of SimpleCov with both MiniTest and RSpec was seamless. We configured only once and every time we run tests, the coverage reports were automatically updated.
2. **CodeClimate:** We used CodeClimate to keep track of code quality issues in our code base. Codeclimate helped us in finding similar pieces of code across multiple Rails controllers or React components. This helped us extract similar code into a single function reducing the total number of lines and increased modularity.
3. **Database_cleaner:** This gem was used to clean the database after every test suite. This helped us make sure that one test did not affect the way other test performed.
4. **React material UI:** We used Material UI's React component library in the frontend of our application. This was of much use to us since all components in this library were by default responsive to various screen sizes. This library also ensured that our design language stayed consistent throughout the application.

Pending Items and Future Outlook:

1. Ability to update password using 'Forgot/Reset Password' feature.
2. Remove selection of user/talent/producer radio buttons when signing up. A regular user who signs up on the web-app should only be taken to the talent dashboard. An admin (event manager) can create other admin and client profiles.

Client profiles with default password can be sent to client via email for them to reset and user - they will be redirected to client dashboard. Admin adding other admins will create a profile taking them to the admin dashboard.

3. Notification icon next to a deck in client view when the producer assigns a new deck to a client.
4. Bug-fix: Removing a talent from the initially curated admin deck (first level of filtering submissions) should remove the talent from the assigned client decks which is being negotiated on UI.
5. Bug-fix: Forms can be re-used by multiple events.
6. Bug-fix: Events should not be created with emp
7. Remove image field in summary table as it currently only displays the file path rather than the actual image.
8. Provide support for previewing PDF uploads. Currently the application supports previewing only image uploads
9. Producer-Client Messaging: A chat interface between an individual producer and client. The backend model to facilitate this is already in-place.
10. Payment: Talent payment post event completion. The summary table can be further extended to support this feature.
11. Mobile View: Optimize the application for viewing on mobile devices.
12. Edit Event – Give the producer the ability to edit a previously created event.
13. Delete Event – Give the producer the ability to delete a previously created event.

Poster Video: https://youtu.be/L_qkOTTMOog

Demo Video: https://youtu.be/_8cd_FA6gkU

Important Links

1. **Pivotal Tracker:** <https://www.pivotaltracker.com/projects/2556130>
2. **Github:** <https://github.com/aakashraj96/CASTNXT>
3. **Heroku Deployment Link:** <https://castnxt-iter5.herokuapp.com/>