COMPONENTS FOR UHHHWUT

UI Components (Generic)

- **<Input>**
  - Stateless
  - Props
    - inputType <String> - specifies what type of input should be rendered, e.g. <input>, <textarea>, <select>
    - isTouched <boolean> - specifies if the user has input anything into this input yet
    - isValid <boolean> - specifies if the input currently in the field is valid
    - label <String> - the display value for the label for this input, if desired
    - all other props that you would normally pass to a <input>, <textarea>, or <select> if you were using them directly… will just be passed down to those HTML elements as {…props}. e.g., value, onChange, className, etc
  - Renders as
    - A <fieldset>, wrapping a <label> (if desired) and the selected inputType
    - Will probably only ever be rendered as a direct child of <Form> probably
- **<Select>**
  - Stateless
  - Props
    - items <array> - array of { displayName, value } objects that will be rendered as the <options> for the select
    - placeholder - will be set as the initial, default value that will be disabled but will visually appear as a prompt
    - all other props that you would normally pass to a <select>
  - Renders as
    - A <select> with <option>s for each item in the items array
    - Will only ever be rendered as a child of <Input>
- **<Form>**
  - Stateless

- Props
  - formConfig <Object> - object with one key for each input, of the form:

    {

        inputType: <String> - inputType of the <Input> to render for this input,

        elementConfig: <Object> - object of config attributes for the underlying HTML input element for this input (e.g., type="email"),

        value: <String> - the value of the input field,

        validation: <Object> - object with properties indicating various validation criteria for this field, must be a validation type implemented in <Form>,

        isTouched: <boolean> - boolean value indicating whether user has input anything into this input yet

    }
  - onSubmit <function> - function to invoke when the form is submitted (internally will first handle e.preventDefault() though)
- Renders as
  - A <form> component that generates <Input> components as specified in formConfig
  - Expects you to wrap buttons as {props.children}, like for a submit button
- Handles
  - onChange for each of the <Input> elements it renders
    - If a given <Input> fires an onChange, it will perform internal validation on the new value for that element according to the rules that were specified in the validation object for that input.
    - It will then call into props.onChange and pass in an object of the form:

      {

          name: <String> - name of the input that changed (e.g., 'email'),

          value: <String> - the new (changed) value of the input,

          isValid: <boolean> - whether or not the new value passed validation,

          isTouched: <boolean> - will always be true because this event will only fire after the user touched the input

      }

    - Basically this just gives me a nice way to quickly create a form with built-in and consistent validation by simply specifying what I want to be in the form and then calling this <Form> component with those specifications
- **<Modal>**
    - Stateless
    - Props
        - isShowing <boolean> - specifies whether or not the Modal is currently being shown
        - onClose <function> - specifies the function to be called when a “close” gesture is performed on the Modal
    - Renders as
        - A <Backdrop> and a wrapping {props.children}
- **<Backdrop>**
    - Stateless
    - Props
        - isShowing <boolean> - specifies whether or not the Backdrop is currently being shown
        - onClick <function> - specifies what to do when you click the backdrop
    - Renders as
        - Just a that spans the full browser window and has some opacity and a z-index above the main content (but below the Modal)
        - Only is ever rendered as a child of the Modal
- **<Card>**
    - Stateless
    - No direct props
    - Renders as
        - A wrapper that specifies card classname stuff, and just wraps {props.children}
- **<Button>**
    - Stateless
    - Props
        - buttonType <String> - can be “success”, “danger”, “back”, “neutralAction”

    - Any other props you'd normally pass to a button will be passed by {…props} as for <Input> (e.g., className, disabled, etc)
- Renders as
    - A <button>, text is in {props.children}

Specific UHHHWUT Components

- **<LoginForm>**
    - Purpose
        - Form accepting email and password inputs and showing a Login button
        - Allows the user to log in to the website
    - State
        - formConfig <Object>: Initial values presented below

key	**inputType**	**elementConfig**	**value**	**validation**	**isTouched**	**isValid**
email	input	{ type: 'text', placeholder: 'Email' }	''	{ required: true}	FALSE	FALSE
password	input	{ type: 'password', placeholder: 'Password' }	''	{ required: true }	FALSE	FALSE

        - didLoginFail <boolean> - set to false initially, will be set to true by handleLoginSubmit if no user with matching credentials was found, and will cause some error to be rendered to the user regarding this
            - Note that didLoginFail is completely independent from form validation conceptually - this is the case where the user has submitted a perfectly valid form as far as the data contained within it is concerned, just the data they submitted didn't happen to map to a real user
    - Functions
        - handleLoginSubmit
            - Handler for submission of login form. Attempt to find a user with matching email and password from API. If found, set user in localStorage, and redirect to '/' (which will now render the app as the user is logged in). Otherwise, show login error message.
        - handleChange

        - Handler for a change in one of the form inputs, will be given an object describing the change from the <Form>
    - Renders as
        - Most crucially a <Form> that is passed its formConfig state as props
- **<RegisterForm>**
    - Purpose
        - Form accepting firstName, lastName, email, password/config password, and nativeLanguage inputs, and showing a Register button
        - Allows the user to register a new account on the website
    - State
        - formConfig <Object>: initial values presented below:

key	**inputType**	**elementConfig**	**value**	**validation**	**isTouched**	**isValid**
firstName	input	{ type: 'text', placeholder: 'First Name' }	”	{ required: true }	FALSE	FALSE
lastName	input	{ type: 'text', placeholder: 'Last Name' }	‘’	{ required: true }	FALSE	FALSE
email	input	{ type: 'text', placeholder: 'Email' }	‘’	{ required: true}	FALSE	FALSE
password	input	{ type: 'password', placeholder: 'Password' }	’'	{ required: true }	FALSE	FALSE
confirmPassword	input	{ type: 'password', placeholder: 'Password' }	‘’	{ required: true, mustMatch: 'password' }	FALSE	FALSE
nativeLanguage	select	{ placeholder: 'Select Your Native Language' }	”	{ required: true }	FALSE	FALSE

    - Functions
        - handleRegisterSubmit

        - Handler for submission of registration form. Will POST a new user to the db on submission, set the user in local storage, and then redirect to '/'.
    - handleChange
        - Handler for an input change, will be given an object with details about the change from <Form>
  - Renders as
    - Most crucially a <Form> that is passed its formConfig state as props
- **<LoginAndRegistration>**
  - Purpose
    - Wrapper for the <LoginForm> and <RegistrationForm> components
    - Allows the user to easily toggle between which form they wish to interact with
    - Handles the logic of assigning classnames to <LoginForm> and <RegistrationForm> components (or wrapper containing them) that will make the whole animation of the registration form concept viable
  - State
    - isLoginMode <boolean> - initially is set to true. Used to determine which form should be showing at any given time - if true show the <LoginForm>, and if false show the <RegistrationForm>
  - Renders as
    - Some sort of wrapper for <LoginForm>, <RegistrationForm> and a <Button> that toggles the isLoginMode state, which on re-render will trigger UI changes in that Button, and in which form is being displayed. The <Button> initially says Create Account when isLoginMode===true, and says "Back to Login" when isLoginMode===false
- **<WelcomeBanner>**
  - Purpose
    - Render the text "Welcome to UHHHWUT" in however many different languages I can come up with
    - Handles the logic for fading out one language and fading in another random language every couple of seconds
    - Note - look at the code you wrote for grimeys website, you very likely want to use something very similar to that here
  - State

        - welcomeMessage <String> - The current welcome message that is being displayed
    - Renders as
        - Probably a <p> or something that just wraps the welcomeMessage from state
- **<UnauthorizedUserLandingPage>**
    - Just a wrapper for <LoginAndRegistration> and <WelcomeBanner>
- **<Header>**
    - Purpose
        - The site header, shows the logo and the title of the page
    - Stateless
    - Renders as
        - A <header> that contains a logo and the title of the page…
- **<Navbar>**
    - Purpose
        - The main navigation bar of the website
    - Stateless
    - Renders as
        - A <nav> that contains a <ul> of <NavLink> components
- **<NavLink>**
    - Purpose
        - A component for a specific link in the navbar
        - Checks location.pathname to see if the current path is “this” link
    - Stateless
    - Renders as
        - A <Link> with an <li> wrapper
        - An extra className of “current” is added to the <li> if location.pathname === props.to
- **<YouTubeSearchBar>**
    - Purpose

        - Allows the user to input a YouTube URL or video ID corresponding to the video that they want to watch in the Theater
    - Stateless
    - Props
        - onChange <function> - function to call when the user types something into the field (after some slight processing first in this component)
        - value <String> - the current value of the input field
    - Handles
        - The logic for determining if the user's input was a URL or a videoId - if URL it will just send the value for the 'v' parameter to parent onChange, and otherwise it will just send the input
    - Renders as
        - An <input> that calls props.onChange on change
- **<TranscriptionRequestControl>**
    - Purpose
        - Allows the user to mark a specific start and end time for a segment they wish to create a transcription for.
    - Stateless
    - Props
        - isRequesting <boolean> - indicates whether or not a transcription request is currently in the process of being recorded / requested
        - onClick <function> - function to call when the user clicks on the button
    - Renders as
        - A <button> (probably not a <Button>) whose label or text (and eventually, icon) will vary based on the isRequesting prop
- **<TranscriptionRequestCard>**
    - Purpose
        - Displays information for a TranscriptionRequest. Conditionally renders stuff based on the state of the TranscriptionRequest passed in.
    - Stateless
    - Props
        - transcriptionRequest <Object> - the transcription request object that this card should display information for, can include an additional transcription

        - onActivate <function> - function to run when the user clicks the “Activate Now” button in the card
        - probably something to indicate like what should be shown - showVideoPreview maybe?
    - Renders as
        - a <Card> that displays information about the TranscriptionRequest, including <Youtube> video preview, start/end time, and buttons, based on the state of the TranscriptionRequest.
            - If !TR.isActive -> render an Activate button
            - if TR.isActive && !TR.transcription -> render maybe like a “Activated - Transcription pending” text node
            - if TR.transcription -> render a View Transcription <Link> disguised as a button
- **<TranscriptionRequestList>**
    - Purpose
        - Renders a list of TranscriptionRequestCards
    - Stateless
    - Props
        - transcriptionRequests <Array> - array of transcription requests to render
        - onActivate <function> (function to call when one of the transcriptionRequests is clicked - just passes arg thru to props.onActivate from that callback)
    - Renders as
        - Some sort of wrapper for <TranscriptionRequestCard>s
- **<TranscriptionRequestWorkshop>**
    - Purpose
        - The view where the user can search for and view a YouTube video, as well as create a transcription request for a specific segment of the video, as well as view all of the relevant TranscriptionRequests they have made for the video.
    - State
        - videoId <String> - the ID of the current YouTube video for the player
        - startTime <number> - the start time of the requested segment, in seconds
        - activatingTranscriptionRequestId <String> - the ID of the transcription request that we are currently editing

    - transcriptionRequests <Array> - array of all transcription requests the user has made for this video
  - Propsless
  - Implements
    - handleYoutubeSearchBarChange
      - Sets videoId to the selected video and startTime and endTime to null. Loads transcriptionRequests from API based on the new videoId and updates transcriptionRequests state array.
    - handleTranscriptionRequestControlClick
      - If startTime is null, sets that. Else, saves a new TranscriptionRequest as { videoId, startTime, currentTimeOfTheYoutubePlayer }, sets startTime to null, and the new TranscriptionRequest db state will update in the state array transcriptionRequests as well
      - Also handles playing the video if paused when user clicks record.
    - handleActivateTranscriptionRequest
      - Sets activatingTranscriptionRequestId state to the ID of the TR we are activating which will render a <TranscriptionRequestActivationWizard> component, passing in the ID of the TranscriptionRequest as prop. Will also set startTime and endTime state to null.
    - handleTranscriptionRequestWizardClose
      - To be called when the Wizard closes (not completed). Just sets the activatingTranscriptionRequestId in state to null
    - handleTranscriptionRequestWizardComplete
      - To be called when the Wizard completes (e.g., user has made it through the entire activation wizard flow and did everything to activate their TR). Might cause a new API poll, depending on if I useContext for TRs/Ts, but will certainly set the activatingTranscriptionRequestId in state to null.
  - Renders as
    - Some wrapper for <YouTubeSearchBar>, <TranscriptionRequestControl>, <TranscriptionRequestCard>, <TranscriptionRequestList>, <Youtube> (from react-youtube), as well as optionally rendering a <TranscriptionRequestActivationWizard> wrapped in <Modal> if isActivating in state is true
- **<TranscriptionRequestConfirm>**
  - Purpose

        - Show you what your transcription request looks like, and render a <TranscriptionRequestForm> if you want to change startTime/endTime.
        - If you change startTime/endTime automatically
    - Stateless
    - Props
        - transcriptionRequest <Object> - All properties of a transcriptionRequest
        - formConfig (from <TranscriptionRequestWizard>, only used to pass it down to <TranscriptionRequestForm>)
        - onChange (for form inputs, just passes the value from <Form> onChange through this function)
    - Renders as
        - A <Youtube> component who will just get its start/end set by incoming transcriptionRequest from props
        - A <Form> that will get formConfig
- **<TranscriptionCreator>**
    - Purpose
        - Show you a segment of a YouTube video from a transcription request, and prompt you to transcribe it
    - Stateless
    - Props
        - transcriptionRequest <Object> - All properties of a transcriptionRequest
        - formConfig (from <TranscriptionRequestWizard>, only used to pass it down to <TranscriptionRequestForm>)
        - onChange (for form inputs, just passes the value from <Form> onChange through this function)
    - Renders as
        - A <Youtube> component who will just get its start/end set by incoming transcriptionRequest from props
        - A <Form> that will get formConfig
- **<TranscriptionRequestActivationWizard>**
    - Purpose
        - The wizard component that guides you through the process of activating your transcription request.

- Basically, a multistep form, that handles the state for both the <TranscriptionRequestForm> and the <TranscriptionForm>, as well as state determining which step in the process you are in.

- State
  - currentStep <number>
    - The step in the process we are in (0 -> confirm my TR, 1 -> transcribe another user's TR, array defining these or enum is defined somewhere)
  - transcriptionRequestToConfirm
    - The transcriptionRequest object loaded from API
    - There is an issue with duplication of information here, as a lot of these properties will wind up on transcriptionRequestFormConfig too… not yet sure the best way to handle this (one way, make the videoId hidden property in the form)
  - transcriptionRequestFormConfig

key	**inputType**	**elementConfig**	**value**	**validation**	**isTouched**	**isValid**
startTime	input	{ type: 'number', placeholder: 'Start Time' }	'', or the startTime value of TR loaded from API w/ id matching from props	{ required: true, mustBeLessThan: 'endTime' }	FALSE	FALSE
endTime	input	{ type: 'number', placeholder: 'End Time' }	'', or the endTime value of TR loaded from API w/ id matching from props	{ required: true, mustBeGreaterThan: 'startTime' }	FALSE	FALSE
language	select	{ placeholder: 'What Language Is This Segment?' }	'', or the language value of TR loaded from API w/ id matching from props	{ required: true }	FALSE	FALSE

- transcriptionRequestToComplete
  - The transcriptionRequest object loaded from API that user must transcribe
- transcriptionFormConfig

key	inputType	elementConfig	value	validation	isTouched	isValid
transcriptionRequestId	input	{ type: 'hidden' }	'', or the id value of TR loaded from API that user must transcribe	{ required: true }	FALSE	FALSE
transcription	textarea	{ placeholder: 'Write transcription here…' }	''	{ required: true }	FALSE	FALSE

- Props
  - transcriptionRequestId
    - Which TR is this the activation wizard for?
  - onFinished
    - Function to call when the wizard is done (functionally this will be used to call a function in parent that holds Modal's isShowing state to set that to false and wipe out the Modal)
- Implements
  - handleTranscriptionRequestChange
    - Handles a change from the transcription request form - updates transcriptionRequestFormConfig for that change event details
  - handleTranscriptionChange
    - Handles a change from the transcription form - updates transcriptionFormConfig for that change event details
  - handleWizardProgressed
    - Handles the user clicking the "next" or "activate" button in either step 1 or 2 of the wizard process.
      - If "next" was clicked, saves the transcriptionRequestFormConfig to API (as PATCH or PUT for the one w/ that ID) and sets currentStep to next step.
      - If "activate" was clicked, saves a new Transcription (POST) based on transcriptionFormConfig and then calls props.onFinished()
  - handleWizardRegressed
    - Handles the user clicking the "back" button in step 2
      - Just sets currentStep to the previous step, maintains all other state.

- Renders as
  - A wrapper that includes a header telling the user which step of the activation process they are in, buttons at the bottom allowing the user to go forward or backward in the process, and in main content either a <TranscriptionRequestConfirm> or a <TranscriptionCreator> component

- **<TranscriptionRequestDashboard>**
  - Purpose
    - View that displays different <TranscriptionRequestList>s based off of properties of those Transcription Requests. A place for the user to see every TR they have ever made, broken up into categories of: those that have new and unapproved Transcriptions created for them, those that are unactivated, those that are activated and awaiting transcription, and those that have been transcribed and accepted.
  - State
    - transcriptionRequests <Array> - array of all TranscriptionRequests for this user
    - activatingTranscriptionRequestId <String> - the ID of the transcription request that we are currently editing
  - Implements
    - handleActivateTranscriptionRequest
      - Sets activatingTranscriptionRequestId state to the ID of the TR we are activating which will render a <TranscriptionRequestActivationWizard> component, passing in the ID of the TranscriptionRequest as prop. Will also set startTime and endTime state to null.
    - handleTranscriptionRequestWizardClose
      - To be called when the Wizard closes (not completed). Just sets the activatingTranscriptionRequestId in state to null
    - handleTranscriptionRequestWizardComplete
      - To be called when the Wizard completes (e.g., user has made it through the entire activation wizard flow and did everything to activate their TR). Might cause a new API poll, depending on if I useContext for TRs/Ts, but will certainly set the activatingTranscriptionRequestId in state to null.
  - Renders as
    - Several different <TranscriptionRequestList>s, one for each state of a TR as specified in the “Purpose” above, as well as a <TranscriptionRequestActivationWizard> that is conditionally rendered based on activatingTranscriptionRequestId state value.

- **<Transcription>**
    - Purpose
        - Show the user the segment that is being transcribed for a transcription, the proposed transcription created by another user, and if the transcription is unaccepted, buttons to either reject or accept the transcription.
    - State
        - transcription <Object> - the transcription loaded from API
    - Propsless
    - Implements
        - handleRejectTranscription
            - Calls function to DELETE this transcription from DB, redirects user to the transcriptions dashboard
        - handleAcceptTranscription
            - Calls function to PATCH true into this transcription's isAccepted property, needs to update transcription state object after this is done
    - Renders as
        - <Youtube> to preview the transcriptionRequest start/end segment, text node for the transcription text, if !transcription.isAccepted -> Reject/Accept <Button>s
- **<UHHHWUT>**
    - Purpose
        - If the user is logged in, render <Header>, <NavBar>, and <ApplicationViews>
        - Otherwise, render <UnauthorizedUserLandingPage>
- **<ApplicationViews>**
    - Handle routing for the app, set up routes for:
        - /workshop
        - /dashboard
        - /transcription/:transcriptionId
        - /logout
    - Set up redirection from / -> /workshop if the user navigates to / while logged in
- **There will also be <Provider> components for each database table**