skenstunts.com - The Readme File

skenestunts.com is the online headquarters for Skene Stunts, a film stunt company based in Western Canada.

The source code for this project lives in a public repository at https://github.com/willrhoda4/skenestunts_production and is hosted at Heroku.

It's supported by the poster_gopher microservice which lives at https://github.com/willrhoda4/poster_gopher and is hosted at Vercel.

This readme documents the project's dependencies, environment variables, database tables and API endpoints, but before we get into all that, let's start with a high-level overview of what the website has to offer...

Description:

The public-facing website includes the following pages:

Info: This is the busiest page on the site, and features the following from top to bottom:

- a bio blurb and team portrait
- Skene Stunts' latest demo reel
- a company values section
- the FAQ buffet
- a simple social media section
- links out to other pages and a customizable quote

Contact: The Contact page provides separate email forms for producers and the public. It also includes a talent form that allows performers to submit themselves to a database, tracking over 120 relevant data points.

Gallery: The Gallery page showcases a photo gallery powered by the Instagram API.

Reels: The Reels page displays a demo-reel gallery with videos embedded from Vimeo.

Media: The Media page is an archive of news stories that the company was featured in.

Team: The Team page includes profiles of each team member.

Update Profile: The Update Profile page allows talent to log in and update their profile information.

The admin dashboard — dubbed Director's Chair — allows the general manager to log in and update content on any of these pages, (besides the gallery, which comes straight from Instagram). The admin also has the power to send invitations to team and board members, which gives them access to a limited version of Director's chair that includes:

- performer search
- access to the poster barn
- the ability to edit their own profile

Pages available to the admin in Director's Chair are as follows:

Misc: Update the background image for the hero header or put the website into construction mode.

Info: Manage the FAQ buffet or update the quote for the info page.

> *NOTE: It's up to you how many FAQs you maintain, but it's important that there always be an FAQ with the id 'advice', because the contact page links to it.*

Reels: Manage the reels for the reels page.

Media: Manage the stories for the media page.

Board: Create and edit profiles for 'board members' – the team members with the big profiles and extra Director's Chair privileges.

Team: Create and edit profiles for 'team members' – the team members with the small profiles and minimal Director's Chair privileges.

Posters: The poster barn is a museum of movie posters from projects the team has worked on. It's helpful for team members picking posters to put on their profiles. It can be updated by pushing the 'update barn' button, which triggers a post request to the poster_gopher.

Performers: This page provides a convenient interface to search through the performer database. Ways to filter are as follows:

- strict boolean conditions (eg: skill or ethnicity searches)
- case-insensitive string conditions (eg: name or email searches)
- numeric range conditions (eg: age or height searches)

Filters can be activated using the checkbox next to them.

Results can be ordered according to one of seven qualities, in ascending or descending order.

At your option, the amount of results can be limited from 1-100.

The envelope icon in the upper-righthand corner opens an email form that'll deliver a message to all search results.

Headshot/bodyshot thumbnails in search results can be momentarily expanded by clicking on them.

Any team member can add a note to a performer, but note that there is only one note per performer, meaning that any team member can overwrite (or add to) that note in the future.

Admin and board members can edit a performer's internal class, team members can just view it.

Clicking 'count credits' sends the poster_gopher to IMDB to see how many credits this performer has.

Clicking on phone numbers and emails copies them to your clipboard.

Clicking on phone or email icons dials the number or opens an email form.

Apart from the performers page and the poster barn, all pages on Director's Chair have a similar set of control buttons. This is how they work:

trash icon: delete item

pen-on-page icon: edit item

up-arrow icon: move item up one position on page

down-arrow icon: move item down one position on page

paper-plane icon: sends out an invitation Director's Chair (team and board section only)

chain icon: copy FAQ link* to clipboard (FAQ section only)

* a link that will take a user straight to the info page and right to the right part of the FAQ buffet

Dependencies:

The following dependencies are required for the project:

```
"axios": "^1.4.0",
"bcrypt": "^5.1.0",
"body-parser": "^1.20.2",
"cors": "^2.8.5",
"dotenv": "^16.0.3",
"express": "^4.18.2",
"googleapis": "^118.0.0",
"multer": "^1.4.5-lts.1",
"nodemailer": "^6.9.2",
"pg": "^8.10.0",
"react": "^18.2.0",
"react-dom": "^18.2.0",
"react-helmet": "^6.1.0",
"react-router-dom": "^6.11.1",
"react-scripts": "5.0.1",
"web-vitals": "^2.1.4"
```

Environment Variables:

The following environment variables need to be configured in production:

REACT_APP_API_URL=	API URL for requests from the front-end
GDRIVE_CREDENTIALS=	Google Drive credentials stored as a string to dynamically render .json
URL=	URL for the client-facing site (used to generate password reset links)
REACT_APP_API_URL=	URL for HTTP requests from client
REACT_APP_LINK_URL=	URL for FAQ links
EMAIL=	Gmail account to handle email needs
EMAILPASS=	Gmail app password
CLIENT_ID=	Google APIs client ID
CLIENT_SECRET=	Google APIs client secret
REDIRECT_URI=	Google APIs redirect URI
REFRESH_TOKEN=	Google APIs refresh token
PERFORMER_IMAGES=	Google Drive ID for performer images folder
TEAM_IMAGES=	Google Drive ID for profile photos folder
BACKGROUND_IMAGES=	Google Drive ID for background images folder
DATABASE_URL=	This variable will automatically be set by Heroku in production, and is automatically updated as account credentials are automatically updated.

If you're not using Heroku Postgres, you may need to use the PG_ variables below in production. These environment variables can be used to connect locally to a database during development (just uncomment the code in database.js):

PG_USER=	Postgres user for local development
PG_HOST=	Postgres host for local development
PG_DATABASE=	Postgres database for local development
PG_PASSWORD=	Postgres password for local development
PG_PORT=	Postgres port for local development

Database:

The skenstunts.com project utilizes a PostgreSQL relational database to store various information related to performers, teams, media, and other website content. All id columns are self-generating.

The database schema includes the following tables:

misc

"misc_id" integer

"description" character varying

"value" character varying

"active" boolean

NOTE: For the site to run properly, misc must maintain rows with the following values for description:

-insta_token

-admin_access

-info_quote

-info_quote_by

-background

-construction_mode

(^^^must also have value false for column active)

faq

"faq_id" integer

"question" character varying

"answer" character varying

"rank" integer

"css_id" character varying

media

"article_id" integer

"headline" character varying

"date" date

"outlet" character varying

“article_url" character varying

"image_url" character varying

"image_description" character varying

"rank" integer

reels

"reel_id" integer

"title" character varying

"caption" character varying

"embed_code" character varying

“rank" integer

performers

"performer_id" (integer)

"first_name" (character varying)

"last_name" (character varying)

"email" (character varying)

"phone" (character varying)

"gender" (character varying)

"pronouns" (character varying)

“workers_union" (character varying)

"imdb_id" (character varying)

"headshot" (character varying)

"bodyshot" (character varying)

"reel_url" (character varying)

"eyes" (character varying)

"hair" (character varying)

"black" (boolean)

"white" (boolean)

"east_asian" (boolean)

"indigenous" (boolean)

"hispanic" (boolean)

"mena" (boolean)

"south_asian" (boolean)

"boxing" (boolean)

"film_fighting" (boolean)

"judo" (boolean)

“jiu_jitsu" (boolean)

"karate" (boolean)

"kung_fu" (boolean)

"mma" (boolean)

"muay_thai" (boolean)

"capoeira" (boolean)

"wrestling" (boolean)

"football" (boolean)

"baseball" (boolean)

"basketball" (boolean)

"cheerleading" (boolean)

"dance" (boolean)

"gymnastics" (boolean)

"hockey_field" (boolean)

"hockey_ice" (boolean)

“hockey_street" (boolean)

“lacrosse" (boolean)

"rugby" (boolean)

"soccer" (boolean)

"softball" (boolean)

"tennis" (boolean)

"volleyball" (boolean)

"bicycle_riding" (boolean)

"bicycle_tricks" (boolean)

"dirt_bike_riding" (boolean)

"dirt_bike_tricks" (boolean)

"horse_bareback" (boolean)

"horse_jousting" (boolean)

"horse_jumping" (boolean)

"horse_riding" (boolean)

"motorcycle_riding" (boolean)

"motorcycle_tricks" (boolean)

"mountain_biking" (boolean)

"precision_driving" (boolean)

"skateboarding" (boolean)

"stunt_driving" (boolean)

“unicycle" (boolean)

"atv_riding" (boolean)

"atv_tricks" (boolean)

"nascar" (boolean)

"canoeing" (boolean)

"high_diving" (boolean)

"jetski_riding" (boolean)

"jetski_tricks" (boolean)

"kayaking" (boolean)

"paddle_boarding" (boolean)

"surfing" (boolean)

"surfskiing" (boolean)

"wakeboarding" (boolean)

"whitewater_kayaking" (boolean)

"whitewater_rafting" (boolean)

"freediving" (boolean)

"occupational_diver" (boolean)

"padi" (boolean)

"skating_ice" (boolean)

"skating_inline" (boolean)

"rollerblading" (boolean)

"skiing_alpine" (boolean)

"skiing_xc" (boolean)

"skiing_downhill" (boolean)

"skiing_freestyle" (boolean)

"skiing_jumping" (boolean)

"mountain_boarding" (boolean)

"snow_biking" (boolean)

"snow_kiting" (boolean)

"snowboarding" (boolean)

"snowmobiling" (boolean)

"air_rams" (boolean)

"archery_horseback" (boolean)

"archery_target" (boolean)

"circus_training" (boolean)

"climbing_ice" (boolean)

“climbing_rock" (boolean)

"descender_work" (boolean)

“fire_burns" (boolean)

"fire_safety" (boolean)

"hang_gliding" (boolean)

"high_falls" (boolean)

"parkour" (boolean)

"prosthetic_work" (boolean)

"skydiving" (boolean)

"slacklining" (boolean)

"stair_falls" (boolean)

"stunt_actor" (boolean)

"trampoline" (boolean)

"wirework" (boolean)

"air_brake" (boolean)

"heavy_trailer" (boolean)

"house_trailer" (boolean)

"class_1" (boolean)

"class_2" (boolean)

"class_3" (boolean)

"class_4" (boolean)

"class_5" (boolean)

"class_6" (boolean)

"krav_maga" (boolean)

"waterskiing" (boolean)

"ice_climbing" (boolean)

"rock_climbing" (boolean)

"tricycle" (boolean)

"rank" (integer)

"birthyear" (integer)

"weight" (integer)

"height" (integer)

"password" (character varying)

"stage_combat" (boolean)

"stage_swordplay" (boolean)

"performer_notes" (character varying)

“admin_notes" (character varying)

"performer_class" (character varying)

"update_count" (integer)

"province" (character varying)

"fencing" (boolean)

"submitted_when" (bigint)

"updated_when" (bigint)

performer_passwords

"password_id" integer

"performer_id" integer <== foreign key references performer_id
to performers and deletes on cascade

"password" character varying

"token" character varying

"reset_at" bigint

board

"team_id" integer

"legal_name" character varying

"imdb_id" character varying

"image_url" character varying

"poster_1" character varying

"poster_2" character varying

"poster_3" character varying

"poster_4" character varying

"poster_5" character varying

"poster_6" character varying

"poster_7" character varying

"poster_8" character varying

"poster_9" character varying

"poster_10" character varying

"profile" character varying

"image_alt" character varying

“rank" integer

"title" character varying

"attribute_1" character varying

“attribute_2" character varying

"attribute_3" character varying

"email" character varying

"uploaded_image" boolean

"publish" boolean

"image_id" character varying

"password" character varying

"no_posters" boolean

board_passwords

"password_id" integer

"team_id" integer <== foreign key references team_id to board
and deletes on cascade

"password" character varying

"token" character varying

"reset_at" bigint

team

"team_id" integer

"legal_name" character varying

"imdb_id" character varying

"image_url" character varying

"poster_1" character varying

"poster_2" character varying

"poster_3" character varying

"poster_4" character varying

"poster_5" character varying

"image_alt" character varying

"rank" integer

"title" character varying

"email" character varying

"uploaded_image" boolean

"publish" boolean

"image_id" character varying

"password" character varying

"no_posters" boolean

team_passwords

"password_id" integer

"team_id" integer <== foreign key references team_id to team
and deletes on cascade

"password" character varying

"token" character varying

"reset_at" bigint

posters

"poster_id" integer

"title" character varying

"imdb_id" character varying

"image_url" character varying

"rank" integer

clips

"video_id" integer

"title" character varying

"caption" character varying

"embed_code" character varying

"rank" integer

resets

reset_id" integer

"token" character varying

"initiated_at" bigint

"type" character varying

API Endpoints:

Below is a list of the available API endpoints along with their descriptions and request body details:

POST /**getData**

Description: Universal getData function that retrieves data from the specified table with optional filters, sorting, and column selection.

Request Body (array):

0: table_name (string): The name of the table to retrieve data from.

1: filter_arrays (array): An array of filter arrays to apply to the query.

2: options (object):

- orderBy (optional string): The column name to order the results by.
- groupBy (optional string): A string specifying the column to group the results by.
- limit (optional number): The maximum number of results to return.
- columns (optional string): A comma-separated string of columns to select. Default is "*".

POST /**deleteData**

Description: Deletes data from the specified table based on the provided parameters.

Request Body (array):

0: table (string): The name of the table to delete data from.

1: pkName (string): The primary key column name.

2: id (string): The primary key of the record to delete.

3: rank (string): The rank of the record to delete.

POST /**reRankData**

Description: Updates the rank of a record in the specified table.

Request Body (array):

0: table (string): The name of the table to update.

1: pkName (string): The primary key column name.

2: id (string): The primary key of the record to update.

3: oldRank (string): The old rank value of the record.

4: newRank (string): The new rank value to assign to the record.

POST /**addData**

Description: Adds new data to the specified table.

Request Body (array):

0: table (string): The name of the table to add data to.

1: columns (array): An array of column names.

2: parameters (array): An array of parameter values.

3: returning (optional string): comma-separated list of column names to return.

PUT /**updateData**

Description: Updates data in the specified table based on the provided conditions.

Request Body (array):

0: table (string): The name of the table to update.

1: columns (array): An array of column names to update.

2: parameters (array): An array of parameter values.

3: conditions (array): An array of condition arrays to filter the update.

```
     eg => [
                [ 'column1', 'targetValue1 ],
                ['column4', 'targetValue2' ]
           ]
```

POST /**newPerformer**

Description: Creates a new performer and inserts their information into the performers and performer_passwords tables. Called once client receives confirmation of successful headshot/bodyshot deposits.

Request Body (array):

0: columns (array): An array of column names.

1: data (array): An array of performer data.

2: password (string): The password for the new performer.

POST /**checkPassword**

Description: Retrieves desired data, including the foreign key value which is used to check the password against the pwTable before returning data to client.

Request Body (array):

0: table (string): The name of the table that stores desired data.

1: email (string): The email address of the user.

2: pwTable (string): The name of the table that stores passwords.

3: pwTableFk (string): The foreign key column name in the password table.

4: password (string): The password to check.

POST /**resetPassword**

Description: Resets the password for a user.

Request Body (array):

0: id (integer): The ID of the user.

1: newPass (string): The new password.

2: table (string): The name of the table that stores the password.

3: fk (string): The foreign key column name in the table.

POST /**newPasswordLogin**

Description: Logs in a user after a password reset.

Request Body (array):

0: table (string): The name of the password table

1: idKey (string): The column name for the user ID.

2: id (integer): The ID of the user.

POST /**registerReset**

Description: Stores a token in the database for a password reset or email invitation response.

Request Body (array):

0: table (string): The name of the password table to store the token.

1: fk (string): The foreign key column name in the table.

2: id (integer): The ID of the user.

POST /**email**

Description: Handles various email-related operations based on the provided type.

Request Body (object):

- **cc** (string): Comma-separated list of email addresses to CC.
- **name** (string): The name of the sender.
- **type** (string): The type of email operation to perform.
- **email** (string): The email address of the recipient.
- **token** (string): The token associated with the email operation (for **password** resets).
- **phone** (string): The phone number of the sender.
- **origin** (string): The location to send users back to after password resets.
- **invite** (boolean): Determines whether password reset was user- or admin-initiated.

- **message** (string): The email message.
- **resetId** (string): The ID for the password reset.
- **subject** (string): The email subject.

POST /**teamPhoto**

Description: Uploads a profile photo to be stored in Google Drive.

Request Body (Form Data object):

- imageUpload (file): The image file to upload.

POST /**background**

Description: Uploads a background image to be stored in Google Drive.

Request Body (Form Data object):

- imageUpload (file): The image file to upload.

POST /**performerPhotos**

Description: Uploads performer photos (headshot and bodyshot) to be stored in Google Drive.

Request Body (Form Data object):

- headshot (file): The headshot image file to upload.
- bodyshot (file): The bodyshot image file to upload.

POST /**newPoster**

Description: Adds a poster to the database.

Request Body (array):

0: title (string): Title of film

1: IMDB ID (string): IMDB ID of film

2: image URL (string): image URL for poster

POST /**getPostersByLetter**

Description: Retrieves posters based on the provided letter.

Request Body (array):

0: letter (string): The letter to filter the posters by.

POST /**getDoublesPosters**

Description: Retrieves posters that have duplicate images.

Request Body (array):

0: table (string): name of poster table

1: column (string): name of id column

2: ids (array): array of IMDB IDs from team member profile

GET /**getPosterList**

Description: Retrieves the complete poster list for select inputs.

GET '/*'

Description: Serves the index.html file for all routes.