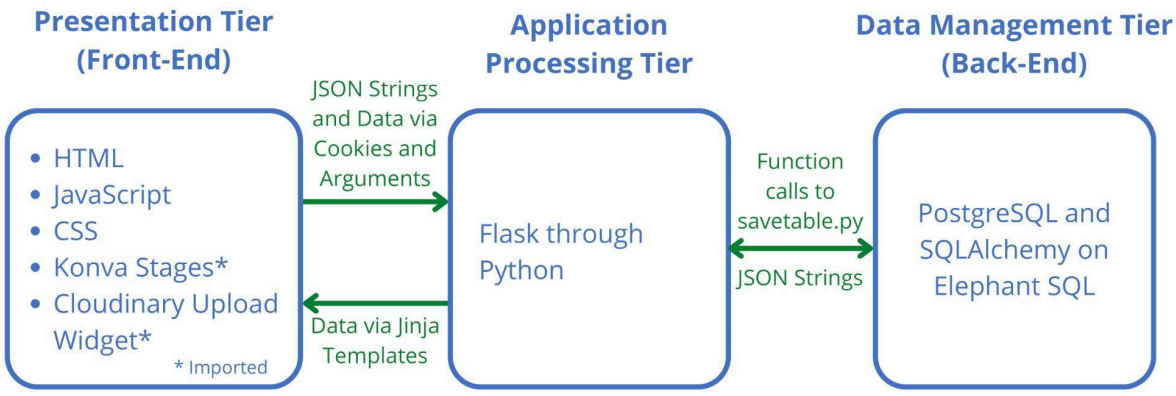
DesignMyDorm Programmer's Guide

By Alice Lee, Brandon Spellman, Sydney Berry, and Advika Srivastava

Three Tier System Software



The front end of DesignMyDorm was written using HTML, JavaScript, and CSS. We also imported the Konva Libraries and the Cloudinary Upload Widget in the front end. The middle tier was written using Flask through Python. Our backend was written using PostgreSQL and SQLAlchemy hosted by Elephant SQL. Communication from the front end to the Processing tier was done by setting arguments and cookies. Communication from the middle tier to the front-end was done via Jinja Templates. The middle tier and back-end communicated via function calls to savetable.py that queries our database hosted by Elephant SQL. The biggest chunk of information we passed through tiers was information regarding furnitures' type, dimensions, and location. We used JSON strings to pass an array containing this information through the processing tier to the back-end. In the back-end, we constructed a JSON string which we then passed through the processing tier to the front-end where we then parsed it to recreate a previous design.

File Tree

├── `auth.py`: This file was written by Alex Halderman, Scott Karlin, Brian Kernighan, Bob Dondero. We use the functions in it to authenticate through CAS.

├── `cert.pem`: This self signed certificate is used to run https locally.

├── `createTable.py`: We ran this file to create the three tables that compose our database.

├── `designmydorm.py`: This is our main flask file. Most of the routes in our flask are explained dramatically in the following pages.

├── `error.html`: This is the html file that gets rendered when our processing tier or backend detects an error.

├── `index.html`: This is the html for our Logged In Design Page. This page is connected to index.js which provides the code for the buttons in this file to work. In addition, index.js turns the container in index.html into our design workspace. Users can also save their design to our database from this page.

├── `indexguest.html`: This is the html file for our Guest Design Page. This page is connected to indexguest.js and works similarly to index.html except users are not able to save their designs.

├── `instructions.html`: This is the html file for our Instructions Page.

├── `key.pem`: This key works with our self signed certificate to run https locally.

├── `landing.html`: This is the html file for our Landing Page (also known as Home). This is the first page users see when they navigate to our website.

├── `load.html`: This is the html file for our Previous Designs Page (also known as See Saved Designs). This page contains buttons to edit, share, and delete designs. Most of these buttons work by constructing URLs based on the room associated with each button that the user navigates to when they press the button.

├── `requirements.txt`: This file includes all the imports are website needs, so render can deploy our app.

├── `runserver.py`: This file is our server, which is connected to flask.

├── `savetable.py`: This file contains the functions we use to query our database. We then return the information we need from each function.

├── `scale.html`: This is the html for the Logged In Scale page. It is where users upload their floor plan through a Cloudinary widget, add Konva transformers, and provide the actual area of their room. This functionally comes from scale.js, which scale.html is connected to.

├── `scaleguest.html`: This is the html for the Guest Scale page. It is very similar to scale.html except that the instructions are not Princeton specific and its navigation features are designed for guests. The html for this file is in scaleguest.js.

├── `static`

│ ├── `bed.png`: This is the graphic for a bed object.

│ ├── `bedclicked.png`: This is the graphic for a bed that is currently the selected piece of furniture.

│ ├── `bookshelf.png`: This is the graphic for a bookshelf object.

│ ├── `bookshelfclicked.png`: This is the graphic for a bookshelf that is currently the selected piece of furniture.

│ ├── `custom.png`: This is the graphic for a custom piece of furniture.

│ ├── `customclicked.png`: This is the graphic for a custom piece of furniture that is currently the selected piece of furniture.

│ ├── `desk.png`: This is the graphic for a desk object.

│ ├── `deskclicked.png`: This is the graphic for a desk that is currently the selected piece of furniture.

│ ├── `dresser.png`: This is the graphic for a dresser object.

│ ├── `dresserclicked.png`: This is the graphic for a dresser that is currently the selected piece of furniture.

│ ├── `favicon.png`: This is the graphic for our favicon.

│ ├── `icon.png`: This is the graphic that is apparent on our Landing (also known as Home) Page. Additionally, it is used as the thumbnails for our Previous Designs (also known as See Saved Designs) Page.

│ ├── `index.css`: This is the CSS file which we use to style our entire website.

│ ├── `index.js`: This is the JavaScript for our Logged In Design Page. It includes the code for all the buttons on our design page and our Konva stage. It includes the code to parse furniture strings to render previously saved designs. It also has the code to create an array with all the furnitures' information when the user saves their design. It also includes code to adjust the stage and all the accompanying furniture to the smaller screen size.

│ ├── `indexguest.js`: This is the JavaScript for our Guest Design Page. It is similar to index.js. However, there is no code to save designs or load previous designs.

│ ├── `landing.js`: This JavaScript file serves as our Cookie Checker. If a user has cookies disabled, it sends a pop up which asks the user to enable cookies.

│ ├── `load.js`: This JavaScript file is connected to load.html and contains the function that is triggered by hitting the “Share” button on our Previous Designs’ Page (also known as See SavedDesigns). It throws a pop up to collect the netid of the person the user wants to share their design with.

│ ├── `one.png`: This is the “1” graphic used in our instructions and scale pages.

│ ├── `scale.js`: This JavaScript file is connected to scale.html, which is our Logged In Scale Page. It provides the code to resize images uploaded to Cloudinary and to display them as a Konva Rect with a pattern fill on our Konva Stage. It also contains the code to create, delete, and bound our Konva Transformers. It also resizes the Konva stage and everything in it when a user resizes their screen. In addition, it calculates the scale value to pass to the Logged In Design page.

│ ├── `scaleguest.js`: This JavaScript is connected to scaleguest.js. The code is almost identical to scale.js except for it passes its calculated scale value to the Guest Design page.

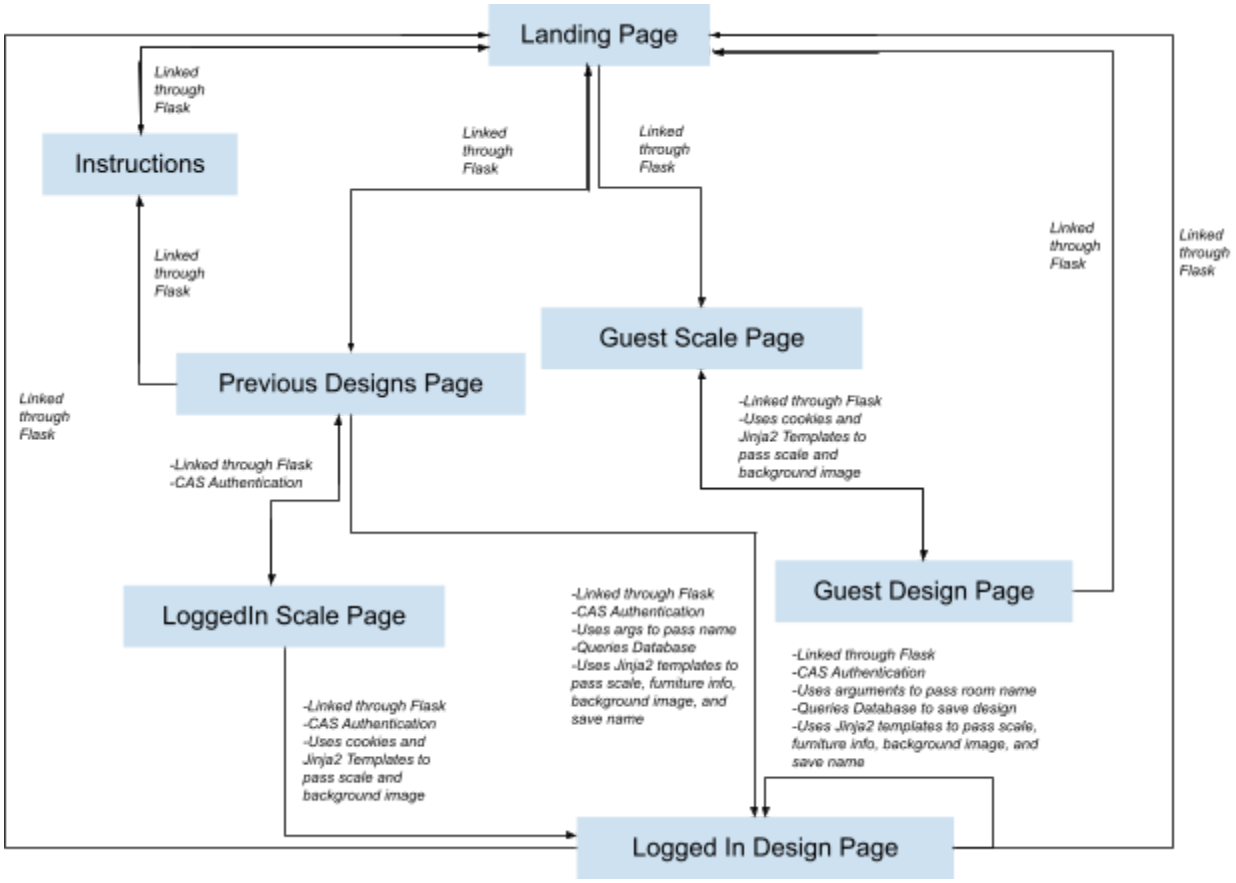
│ ├── `three.png`: This is the “3” graphic used in our instructions and scale pages.

│ ├── `two.png`: This is the “2” graphic used in our instructions and scale pages.

│ ├── `wardrobe.png`: This is the graphic for a wardrobe object.

│ └── `wardrobeclicked.png`: This is the graphic for a wardrobe that is currently the selected piece of furniture.

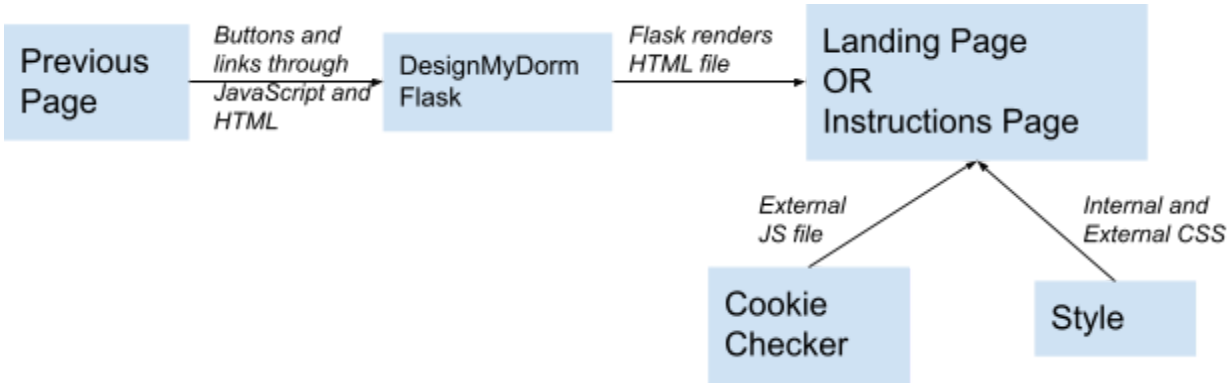
Connection between HTML Pages



This diagram shows the high level flow through our website. Every arrow means there is a link or button on the first page to get to the second page. How our program handles getting from the first page to the second page is explained in more detail the diagrams on the following pages.

Rendering Landing and Instruction Pages

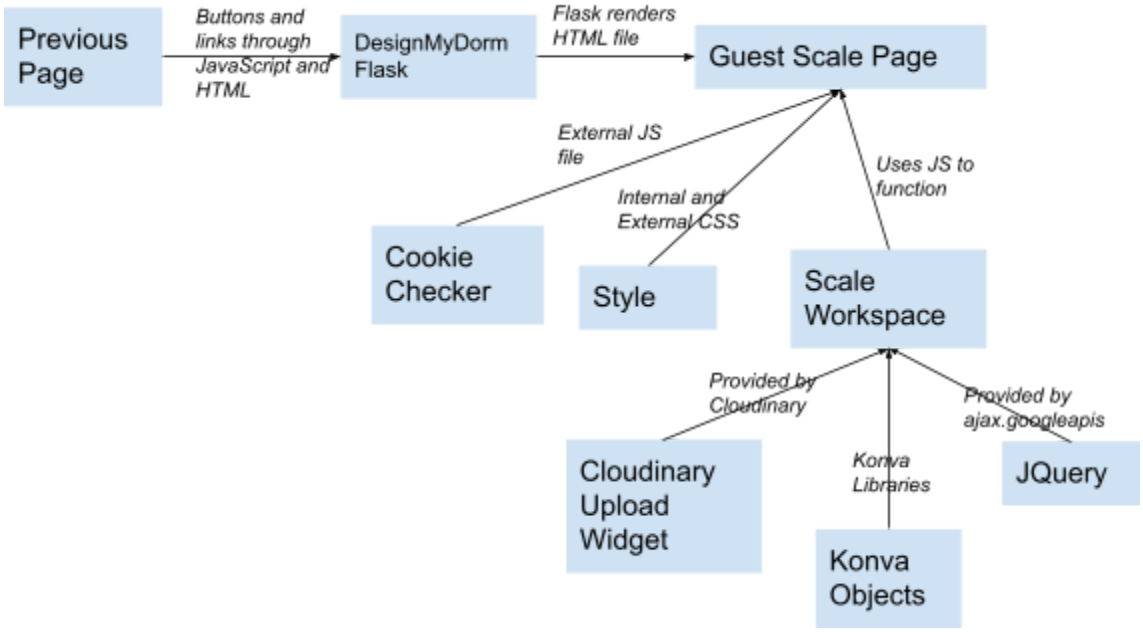
DesignMyDorm functions: /home and /instructions



Our Landing and Instructions pages don't require any information from a previous page or from our database. Users can navigate to these pages through URLs, buttons, or the links within our navigation bar. When users navigate to these pages from a previous page, designmydorm.py (our flask file) renders the resulting html page. Our external CSS file is connected to both of these pages to determine the style of these pages. There is one external cookie checker JavaScript file connected to both of these files. Our cookie checker JavaScript was modified from https://stackoverflow.com/questions/4603289/how-to-detect-that-javascript-and-or-cookies-are-disabled. The original code worked well for Chrome, but we had to add the code for it to work with Safari. If cookies are not enabled, the JavaScript file throws a pop-up to encourage the user to enable cookies.

Rendering Guest Scale Page

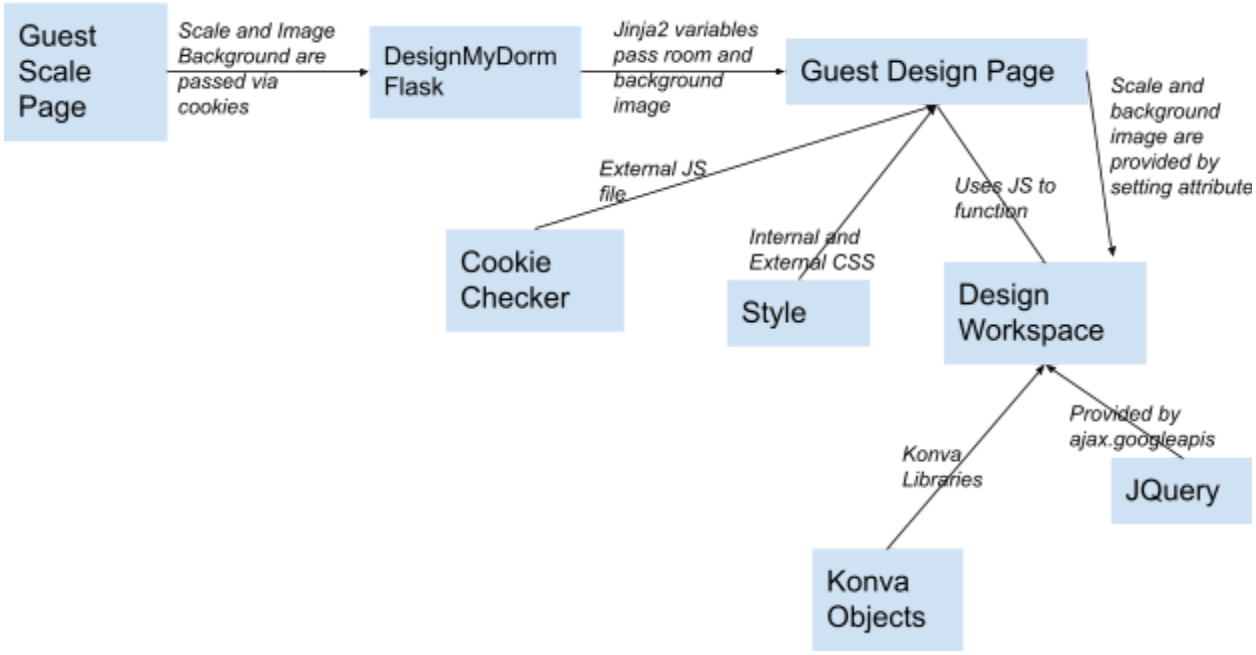
DesignMyDorm function: /scaleguest



Our Guest Scale Page does not require any information from a previous page or from our database. Users can navigate to these pages through URLs, buttons, or the links within our navigation bar. This Guest Scale page has the same external CSS file and cookie checker JavaScript file connected to it as the landing and instructions pages. In addition, the Guest Scale Page has a JavaScript file attached to it (scaleguest.js) that contains the code for the inner workings of the buttons and design stage for this page. This JavaScript file uses three three imported external sources. The file uses some JQuery which was provided by ajax.googleapis. The Cloudinary upload widget provided by Cloudinary along with code from https://docs.google.com/document/d/1VIAlGt38dkor9m2LiOwQ1X5fBkRYVDoNJVK1py_W5jc/edit are used to save the uploaded images to Cloudinary. The uploaded images are modified using a canvas to resize before making a Konva image and adding it to the stage. There are occasional issues with this external file beyond our scope. In this case, there is a ReferenceError and our code throws an alert to the user to reload the website. Our stage along with the Konva Image object and Konva transformer utilize the Konva Libraries. Documentation for Konva transformers can be found at https://konvajs.org/api/Konva.Transformer.html. We used provided functions to help write the code for the “Add Box” and “Delete Box” buttons. We wrote code expanding these libraries to change the color of the box when a user clicks, drags, or, transforms a box. Additionally, we wrote bounding equations to keep the transformers within the stage as well as code to resize the stage and the objects in it when the user resizes their screen. (The design decisions around this are included in our “design problems we encountered” section).

Rendering Guest Design Page

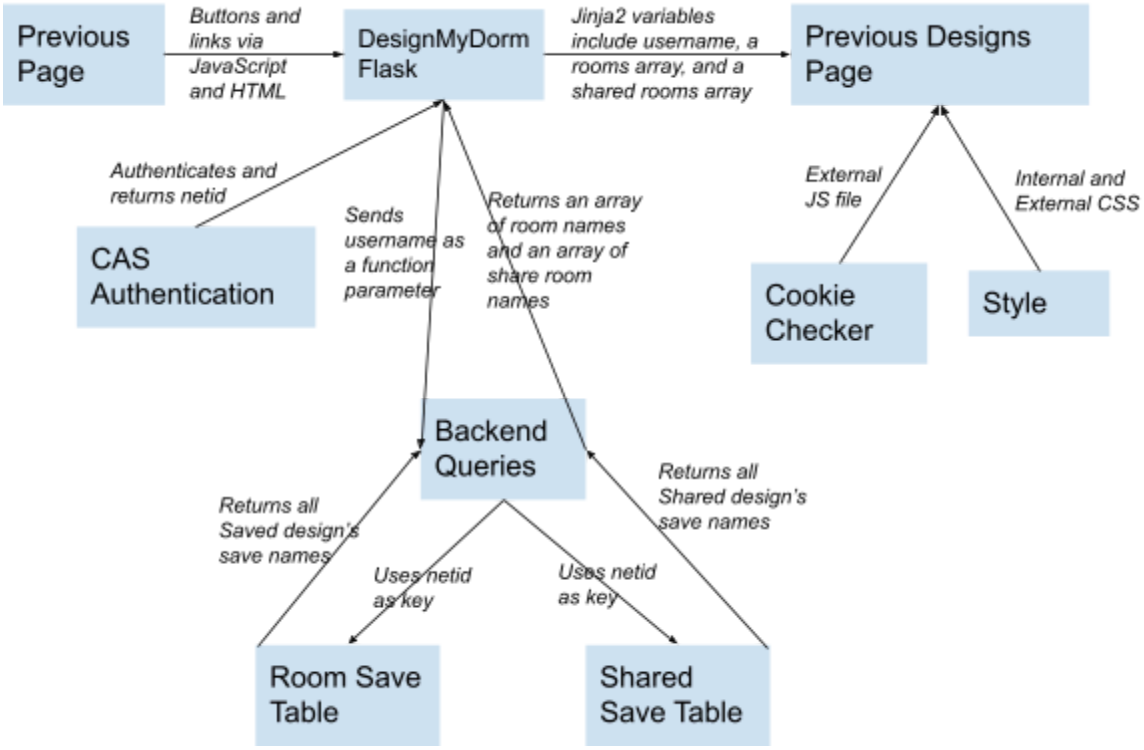
DesignMyDorm function: /designguest



When a user presses the “start designing button,” the Guest Scale Page calculates a scale factor using the pixel area of the boxes currently on the screen and the actual area provided in a text field by the user. This scale value along with the url for the user’s uploaded background image (a url to a white image is used if the user did not upload a background image) are set in cookies. designmydorm.py then renders a Jinja2 template including the url to the background image and the scale value. The background image url and scale value are then provided to designguest.js by setting and getting document attributes. Similarly to scaleguest.js, designguest.js uses and expands on Konva Rects, Images, Stages, and Layers to create the design workspace. For this file, we use graphics we designed and stored in the static file and resize using canvases as pattern fills inside Konva Rect objects. Additionally, we wrote bounding equations to keep the transformers within the stage as well as code to resize the stage and the objects in it when the user resizes their screen. (The design decisions around this are included in our design problems we encountered section). We also added an export to png button that modifies code from this tutorial https://konvajs.org/docs/data_and_serialization/High-Quality-Export.html. The Guest Design page has the same external CSS file and cookie checker JS file connected to it as the landing and instructions pages.

Rendering Previous Design Page

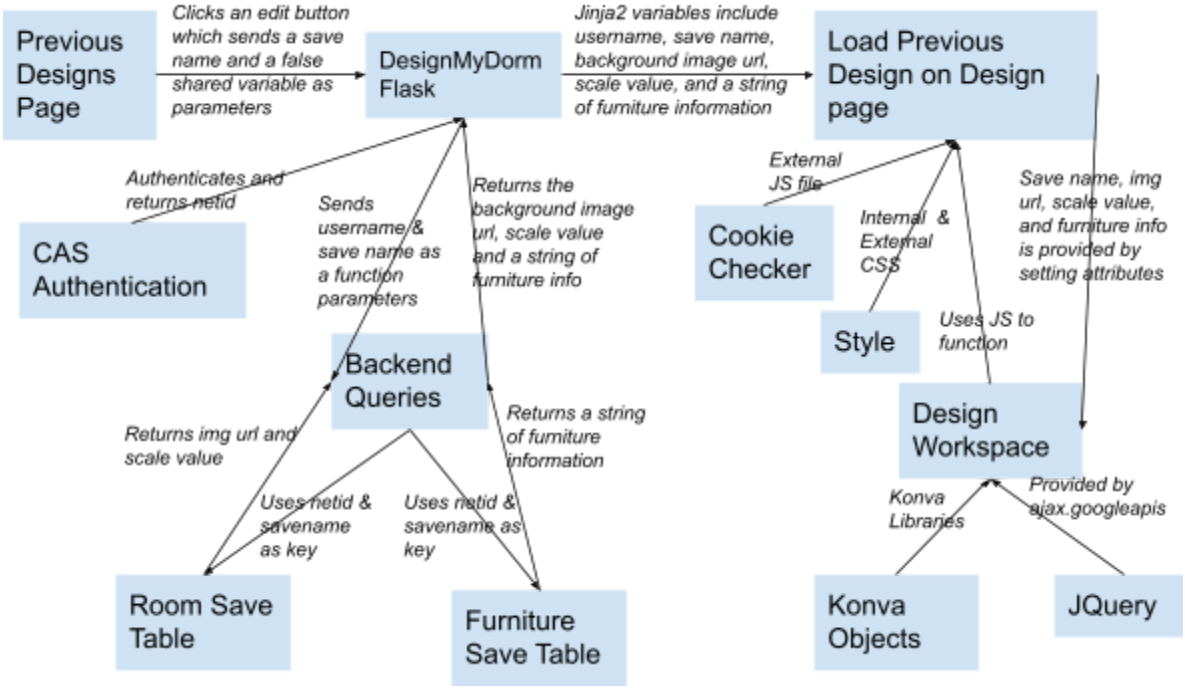
DesignMyDorm Function: /initload



When a user first logins or clicks on “See Saved Designs”, they are redirected to the initload page which shows all designs that was saved by the user using their NetID or was shared with them by another Princeton undergraduate. designmydorm.py uses CAS authentication to authenticate the user and return the user’s netid. designmydorm.py then sends the netid to savetable.py as a function parameter. savetable.py then queries the room_save and shared_save tables using the user’s netid as the key to find the save names of all the rooms the user has saved and all the rooms that have been shared with the user respectively. These names are returned in arrays to designmydorm.py, which provided these arrays along with the user’s netid to the Previous Designs Page (load.html) via a Jinja2 template. The Previous Designs’ page has the same external CSS file and cookie checker JS file connected to it as the landing and instructions pages.

Editing a Previous Design

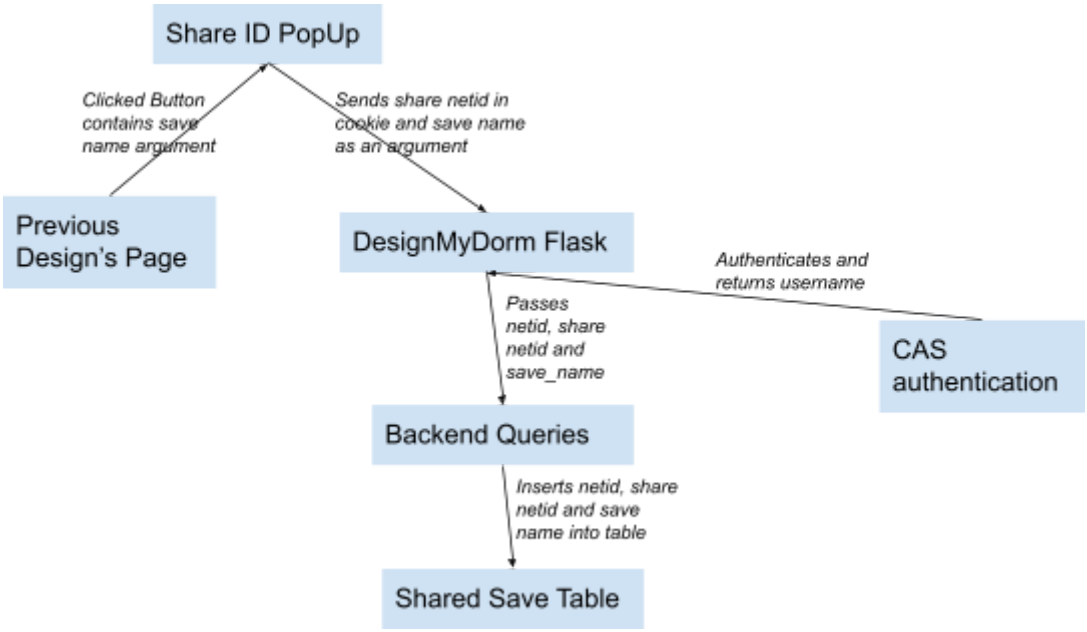
DesignMyDorm Function: /load?save_name=XXXXXX&shared=False



When the user clicks an “Edit” button for a saved design on the Previous Designs’ page, the resulting url includes the save name of the design the “Edit” button was associated with and a “False” shared value to indicate that this wasn’t a shared design. designmydorm.py gets these arguments. designmydorm.py uses CAS authentication to authenticate the user and return the user’s netid. Designmydorm.py then sends the save name and the user’s netid to savetable.py as a function parameter. savetable.py queries the room_save table using the save name and netid to retrieve the room’s scale value and the url for the room’s background image. savetable.py queries the furniture_save table using the save name and netid to retrieve a string containing the type, width, height, rotation, x_coord, and y_coord of every piece of furniture in the room. The background image url, scale value, and furniture string are then returned to designmydorm.py, which sends these along with the save name and user’s netid in a Jinja2 Template to render the previous design on a Logged In Design page (index.html). The image url, scale value, furniture string and savename are passed to index.js to recreate the stage of the room design. This involves parsing the furniture string to regenerate all the furniture as Konva Rects with fillPatterns of graphics. After the page is recreated, the functions in index.js function the same as the do for any design on an index.html page (see *Rendering Logged In Design Page from Scale Page* for more information). The save name is used to construct a string to be the title of an exported PNG.

Sharing a Previous Design

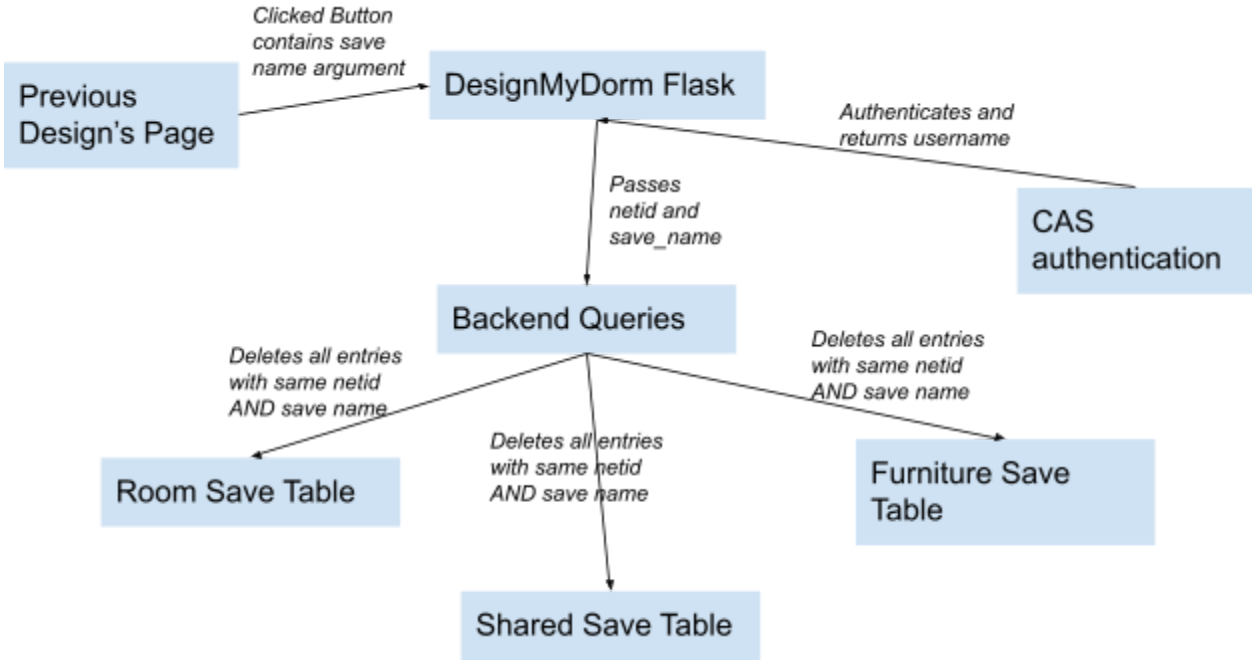
DesignMyDorm Function: /share?sharename=XXXXXX



When the user clicks a "Share" button for a saved design on the Previous Designs' page, the button's value (a url containing the save name as a key value pair) is passed to a function in load.js. This function uses a pop up to collect the netid of the person the user wants to share the design with. The share netid is then set as a cookie before our program routes to designmydorm.py using the url which was the button's value. This setup was necessary to get around parsing issues with save names that contain apostrophes that we originally tried to send as a function parameter. designmydorm.py uses CAS authentication to authenticate the user and return the user's netid. designmydorm.py then passes the user's netid, the netid they are sharing the design with, and the save name to savetable.py. savetable.py queries the share_save table and inserts the netid, the shared netid, and the save name in a row of the table. After the design is shared, our program renders and displays the Previous Designs' Page for more information on this process see *Rendering Previous Design Page*.

Deleting a Previous Design

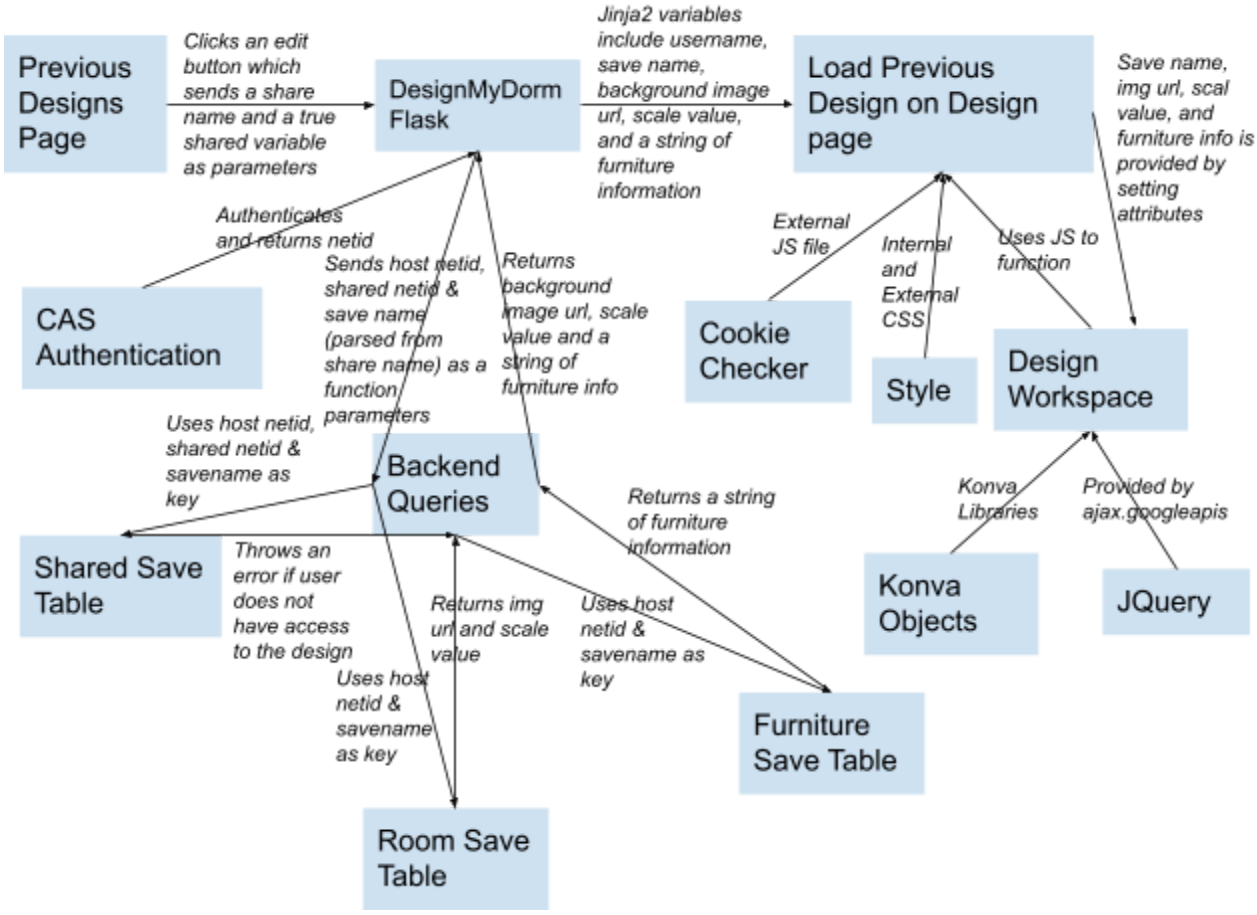
DesignMyDorm Function: /delete?save_name=XXXXXX



When the user clicks a "Delete" button for a saved design on the Previous Designs' page, the resulting url includes the save name of the design the "Delete" button was associated with. designmydorm.py gets these arguments. designmydorm.py uses CAS authentication to authenticate the user and returns the user's netid. Designmydorm.py then sends the save name and the user's netid to savetable.py as a function parameter. savetable.py queries the room_save, furniture_save, and shared_save tables using the save name and netid to delete any rows where the save name and netid matches the query parameters. We have database-related error-handling in this method that redirects the user to the error page in case something goes wrong while querying the database. After the design is deleted, our program renders and displays the Previous Designs' Page for more information on this process see *Rendering Previous Design Page*.

Editing a Shared Design

DesignMyDorm Function: /load?save_name=XXXXXX&shared=True

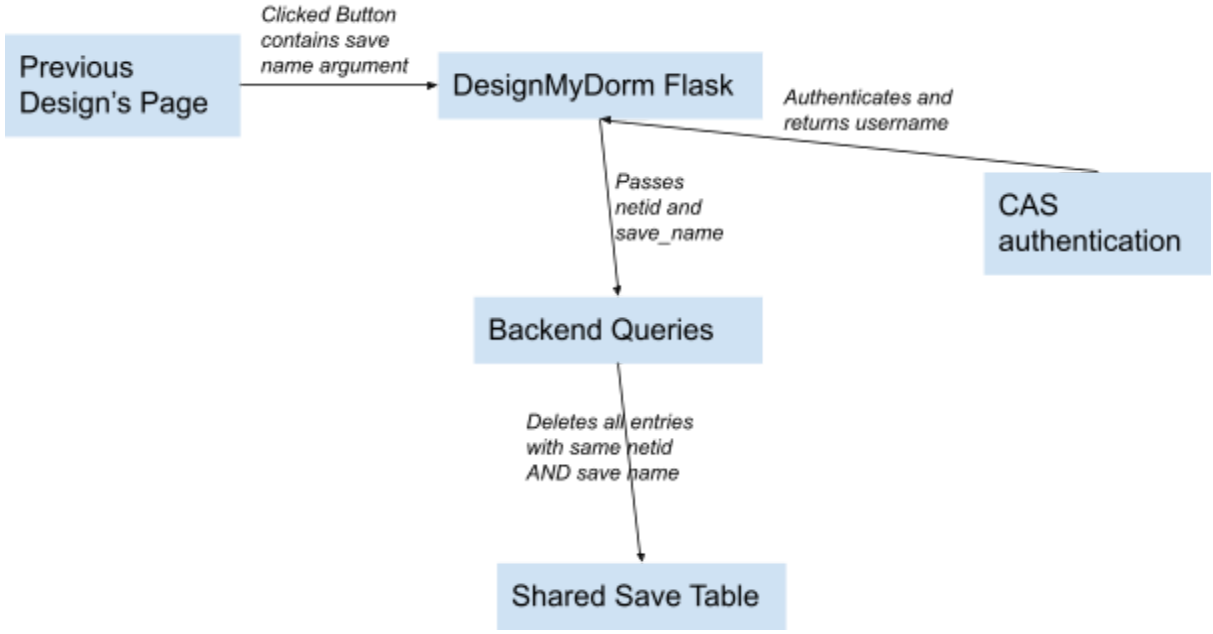


When the user clicks an “Edit” button for a shared design on the Previous Designs’ page, the resulting url includes the share name of the design the “Edit” button was associated with and a “True” shared value to indicate that this wasn’t a shared design. designmydorm.py gets these arguments. designmydorm.py uses CAS authentication to authenticate the user and return the user’s netid. The “True” shared value tells designmydorm.py to parse the share name to get the owner’s netid and the save name for the design. Designmydorm.py then sends the save name and the sharer’s netid (host netid) and the user’s netid (shared netid) to savetable.py as function parameters. savetable.py quires shared save table using the netids and savename to make sure the design has been saved with the user. savetable.py queries the room_save table using the save name and sharer’s netid to retrieve the room’s scale value and the url for the room’s background image. savetable.py queries the furniture_save table using the save name and sharer’s netid to retrieve a string containing the type, width, height, rotation, x_coord, and y_coord of every piece of furniture in the room. The background image url, scale value, and furniture string are then returned to designmydorm.py, which sends these along with the save

name and user’s netid in a Jinja2 Template to render the previous design on a Logged In Design page (index.html). The image url, scale value, furniture string and savename are passed to index.js to recreate the stage of the room design. This involves parsing the furniture string to regenerate all the furniture as Konva Rects with fillPatterns of graphics. After the page is recreated, the functions in index.js function the same as the do for any design on an index.html page (see *Rendering Logged In Design Page from Scale Page* for more information). The save name is used to construct a string to be the title of an exported PNG.

Deleting a Shared Design

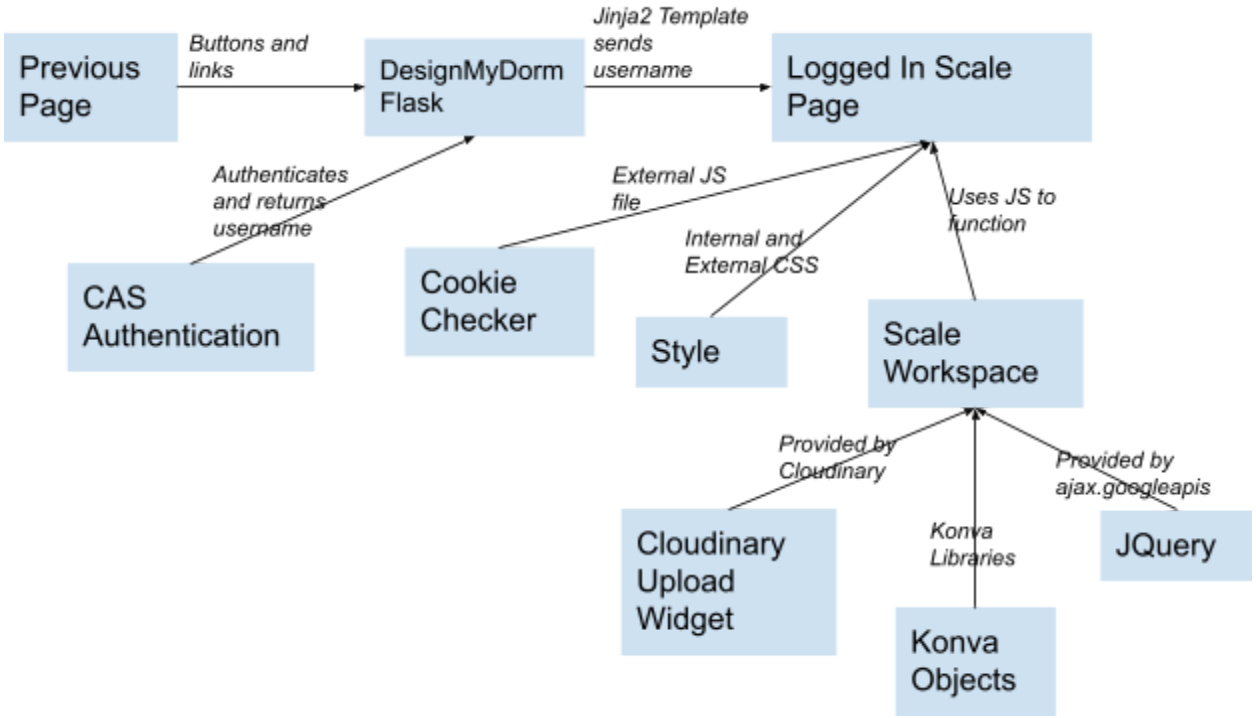
DesignMyDorm Function: /sharedelete?save_name=XXXXXX



When the user clicks a "Delete" button for a shared design on the Previous Designs' page, the resulting url includes the save name of the design the "Delete" button was associated with. designmydorm.py gets these arguments. designmydorm.py uses CAS authentication to authenticate the user and returns the user's netid. Designmydorm.py then sends the save name and the user's netid to savetable.py as a function parameter. savetable.py queries the room_save, furniture_save, and shared_save tables using the save name and netid to delete any rows where the save name and netid matches the query parameters. We have database-related error-handling in this method that redirects the user to the error page in case something goes wrong while querying the database.

Rendering Logged In Scale Page

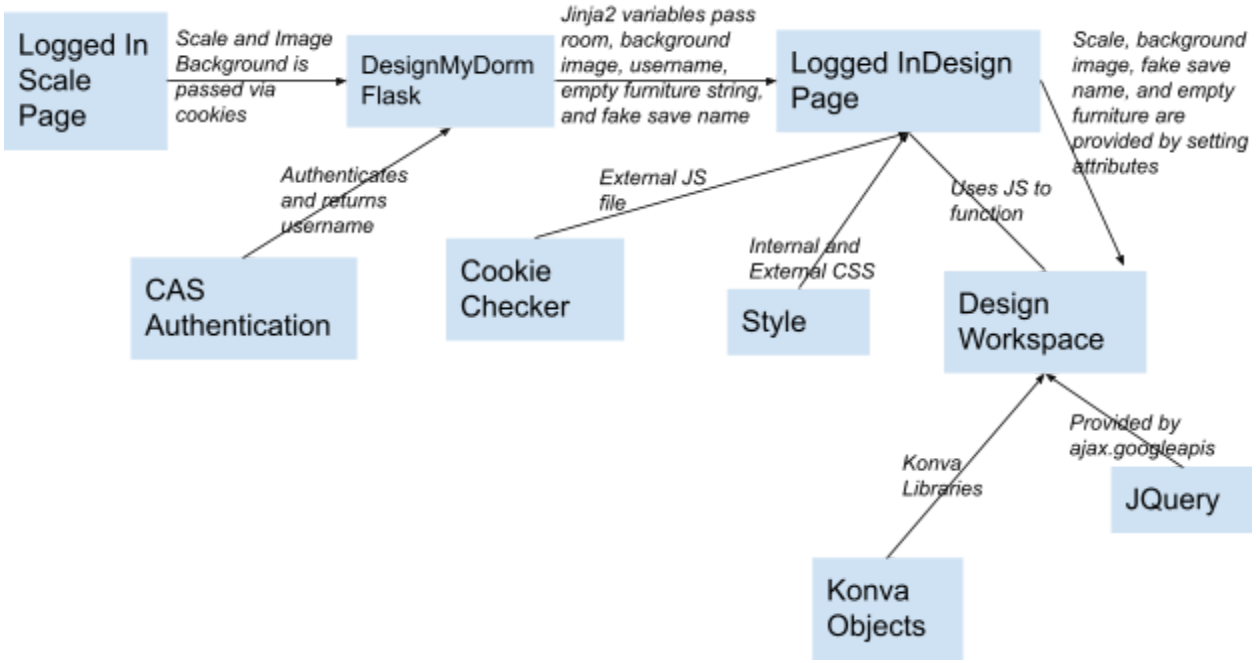
DesignMyDorm Function: /scale



Rendering the Logged In Scale Page is very similar to *Rendering Guest Scale Page*. The only difference is designmydorm.py uses CAS authentication to authenticate the user and returns the user's netid. The netid is then passed to the scale page using Jinja2 templates. The logged in scale page uses scale.html and scale.js. These are very similar to the guest code. We just needed two files to make sure the files redirected to the proper logged in design and the guest design page.

Rendering Logged In Design Page from Scale Page

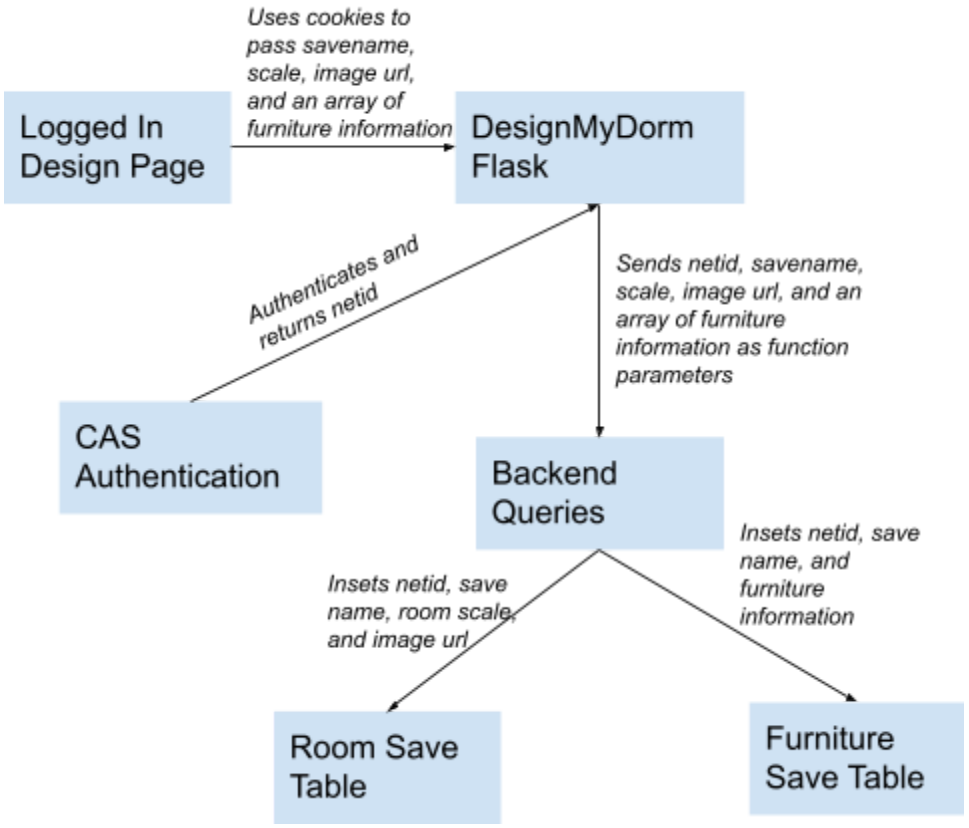
DesignMyDorm Function: /design



Rendering the Logged In Design Page is very similar to *Rendering Guest Design Page*. One difference is designmydorm.py uses CAS authentication to authenticate the user and returns the user's netid. The netid is then passed along with the url to the background image, the scale value, an empty furniture string, and "DesignMyDorm" as a fake save name to index.html using Jinja2 templates. This page then renders similarly to a page of a Previous Design rendering accept there is no furniture in the string to load. For more information see *Editing a Previous Design*.

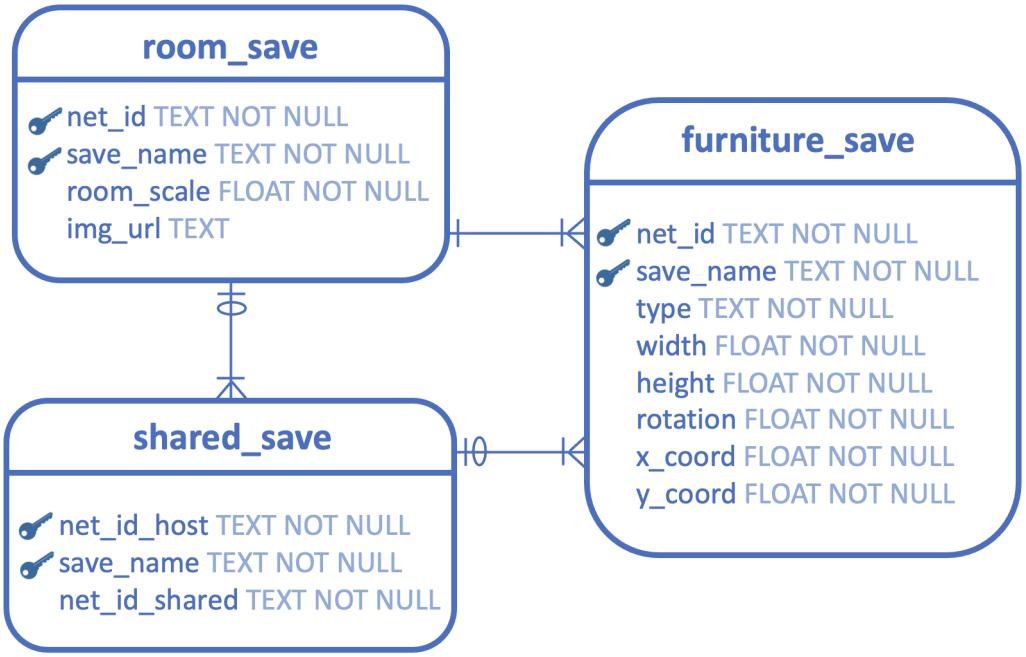
Saving a Design

DesignMyDorm Function: /save



When the user clicks the “Save This Design” after entering a valid save name button on the Logged In Design page, index.js sets cookies for savename, scale, background image, and an array of furniture information. The furniture array includes the furniture type, dimensions, and location information. designmydorm.py uses CAS authentication to authenticate the user and returns the user’s netid. designmydorm.py sends the netid, savename, scale, image url, and the furniture information array as function parameters to savetable.py. savetable.py then queries the room_save table to insert a row containing the netid, save name, room scale, and background image url. Savetable.py parses the furniture array and inserts each piece of furnitures' information as a row in the furniture_save table with the save name and netid as keys. After saving a design, our website reloads the design using the same process that loads a previous design. For more information on reloading, see *Editing a Previous Design.*

DataBase Schema



Our database has 3 tables. And all are keyed by the users NetID and the name of the save. Our main table is room_save. Every row in this table is its own dorm room save. In this table we also have the floor plan background image and the room scale information.

The next table is furniture_save. Each row in this table is a piece of furniture, and has the furniture type, x and y positions, and rotation. Once again, this table is also keyed by the users NetID and the name of the room save it belongs too.

Our third table is shared_save. Each row in this table connects a NetID and room name to the NetID of the user it is shared with. This table allows us to fetch room_save information and furniture_save information for the shared user.

Error Handling

Front-End: Our program aims to catch as many errors as soon as possible. We handle front end errors through alert messages. This allows for our website to preserve the state of the page as it was right before the error was detected.

Errors we detect in the front end include:

- ☐ Having Cookies Disabled
- ☐ A warning to refresh the page if the external Cloudinary Upload Widget did not load properly
- ☐ Not having any boxes on a scale page when the user clicks “Start Designing”
- ☐ Not entering a number greater than 0 in “Actual Area” before the user clicks “Start Designing”
- ☐ Having the scale page proceed to the design page with an unreasonably large scale that would make the autoscaled furniture in our design page not functionable
- ☐ Not entering a number greater than 0 for width or height before clicking “Add Custom Furniture”
- ☐ Entering dimensions for custom furniture that is unreasonably large and would not function properly in our design interface
- ☐ There is a warning on the “Delete All Furniture” button to make sure the user meant to click this button.
- ☐ Having an empty save name
- ☐ Having a save name longer than 15 characters
- ☐ Having a save name that does not contain an ASCII character

Processing-Tier: When our processing tier detects an error, it renders error.html which contains an error message.

Errors we detect in the front end include:

- ☐ A cookie or get argument request does not return a value
- ☐ A savename that will be saved in our room_save and furniture_save table is empty
- ☐ A savename that will be saved in our room_save and furniture_save table contains more than 15 characters
- ☐ A savename that will be saved in our room_save and furniture_save table contains a character that is not ASCII
- ☐ A shared design does not contain an underscore, which is required based on our formatting conventions
- ☐ Our load functions’ “shared” value is not either “True” or “False”.
- ☐ An error occurred in the back-end and needs to be elevated to the front-end

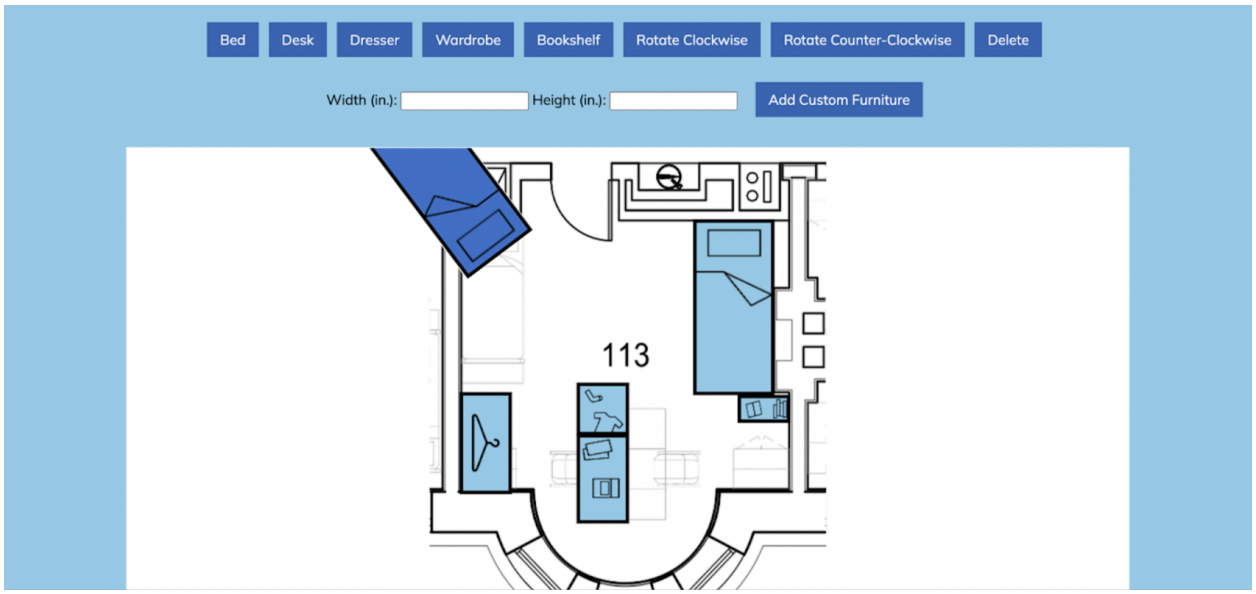
Back-End: We have incorporated error-handling in the backend, and we communicate errors with the frontend by using booleans. When an error is detected in the backend, it returns “True” for the err variable and an error message. If err is “True,” the processing tier renders error.html containing the error message from the back-end.

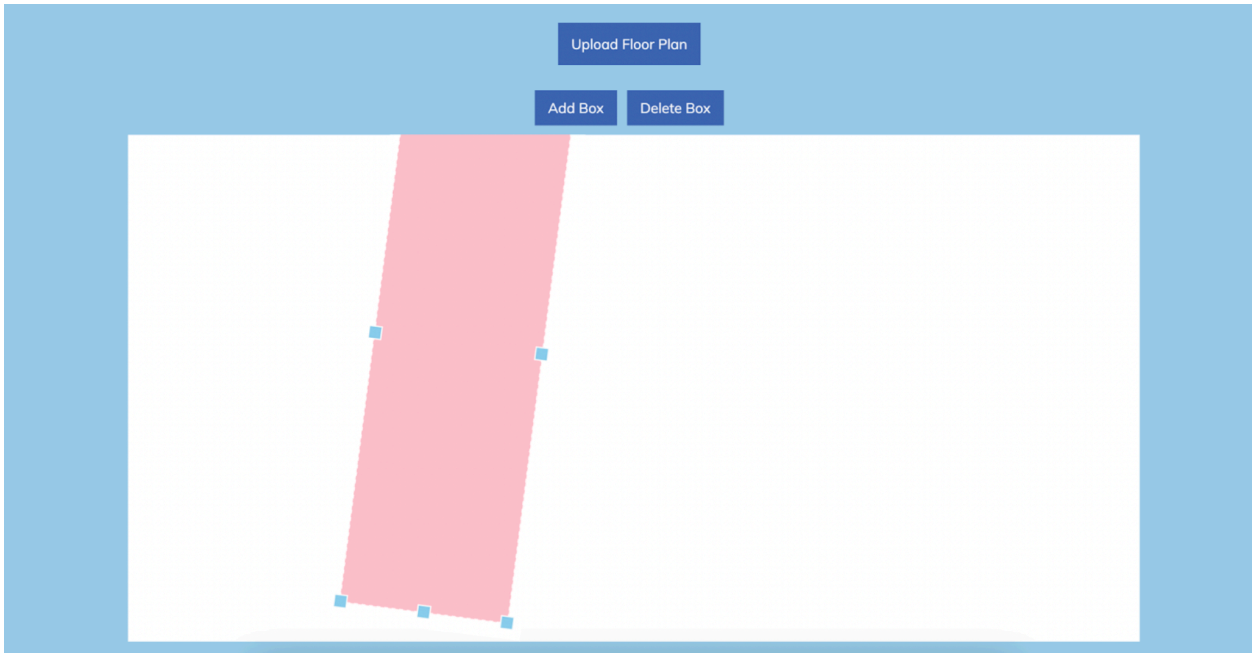
Errors we detect in the Back-end:

- ☐ Attempting to access a design that was not shared with you by manually typing in a share url to match our naming conventions.
- ☐ Attempting to load a design that has already been deleted (maybe in a different tab after the Save Designs Page was loaded)
- ☐ Trying to access a design that does not exist
- ☐ Database being corrupted or not opening

Design Challenges

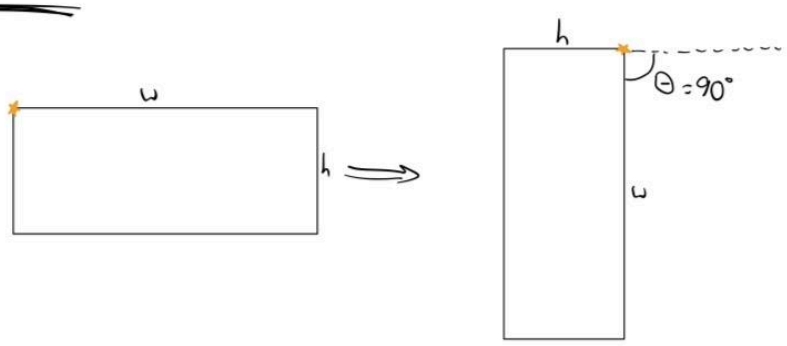
Bounding Konva Rects and Transformers: We used Konva's draggable objects on a Konva stage as our workspaces. However, Konva did not bind their objects to their stage. Thus, users could drag the objects out of the work area.



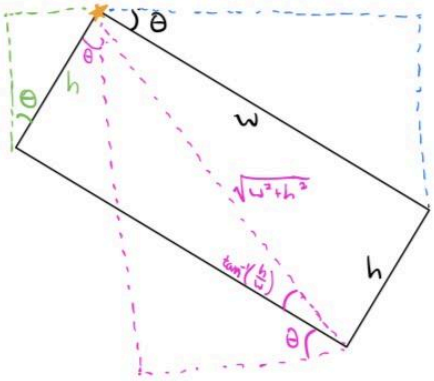


To solve this problem, we had to use trigonometry. We first started by sketching out the bounding equations that we would need.

$0° \rightarrow 90°$



left: $h \cdot \sin(\theta)$ right: $w \cdot \cos(\theta)$



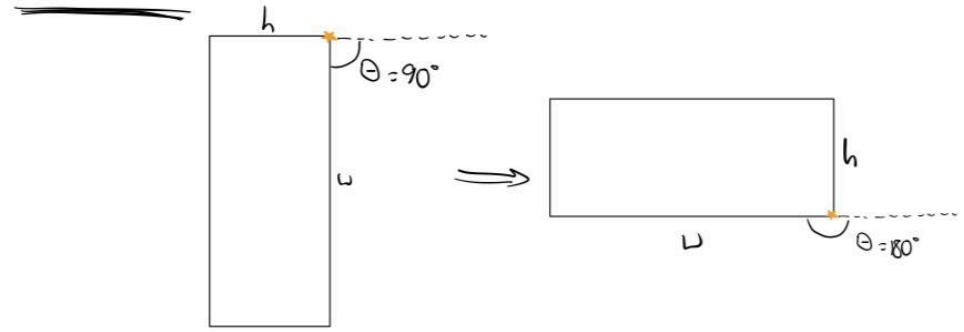
Top Bound : 0

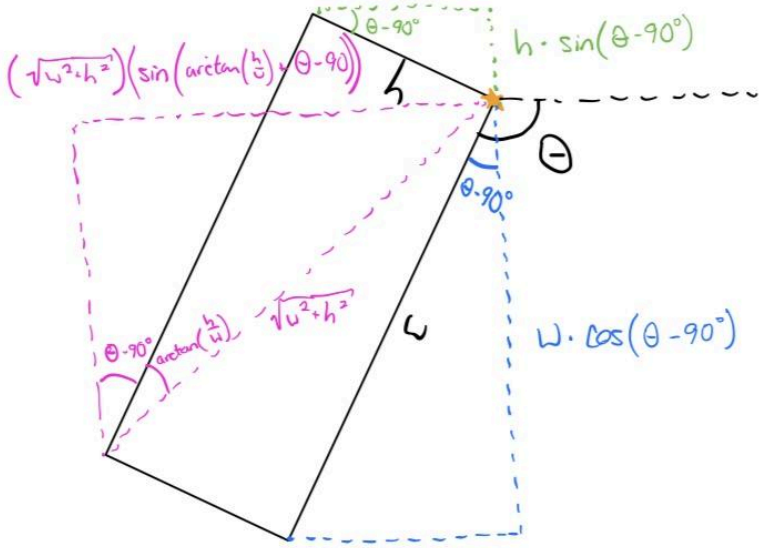
Bottom Bound : $\left(\sqrt{w^2+h^2}\right) \cdot \sin\left(\theta + \arctan\left(\frac{h}{w}\right)\right)$

Left Bound : $h \cdot \sin(\theta)$

Right Bound : $w \cdot \cos(\theta)$

90° → 180°





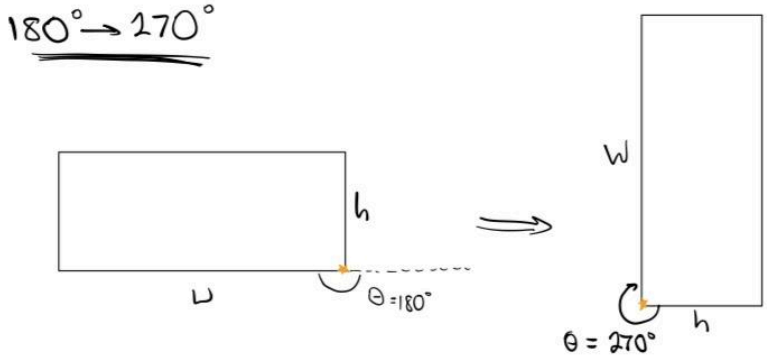
Top Bound : $h \cdot \sin(\theta - 90^\circ)$

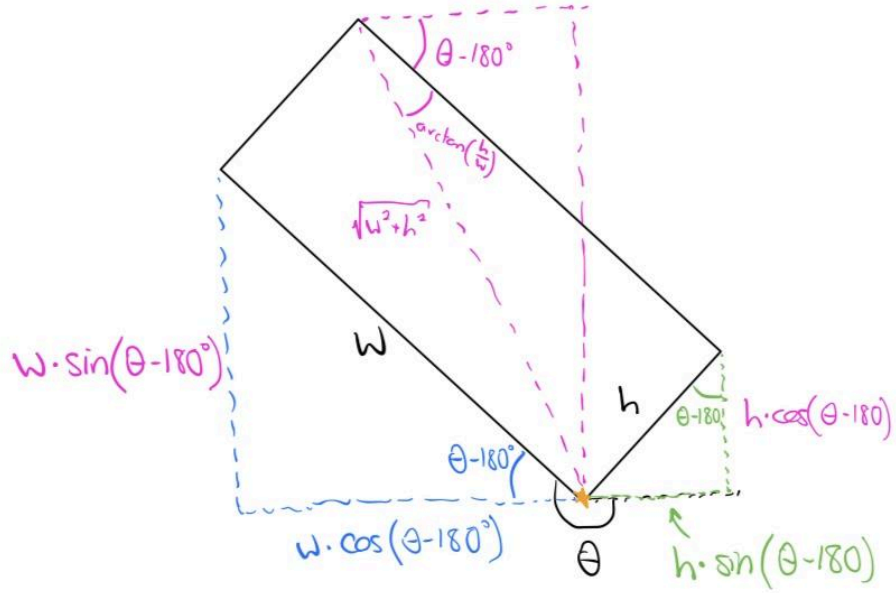
Bottom Bound : $w \cdot \cos(\theta - 90^\circ)$

Left Bound : $\left(\sqrt{w^2+h^2}\right)\left(\sin\left(\arctan\left(\frac{h}{w}\right)+\theta-90\right)\right)$

Right Bound : 0

180° → 270°





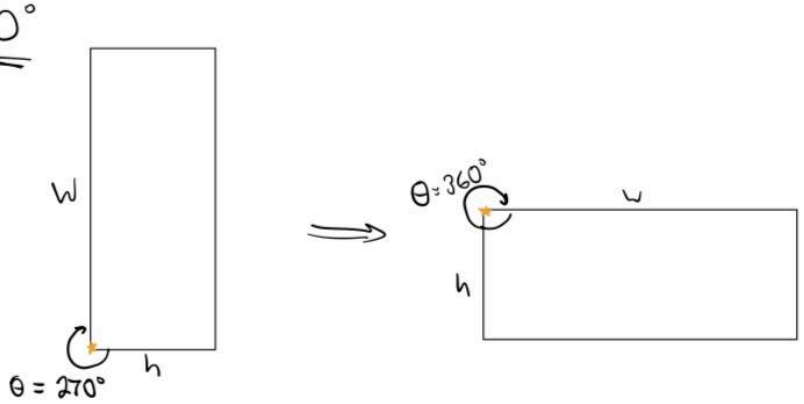
Top Bound : $h\cdot\cos(\theta-180) + w\cdot\sin(\theta-180^\circ)$

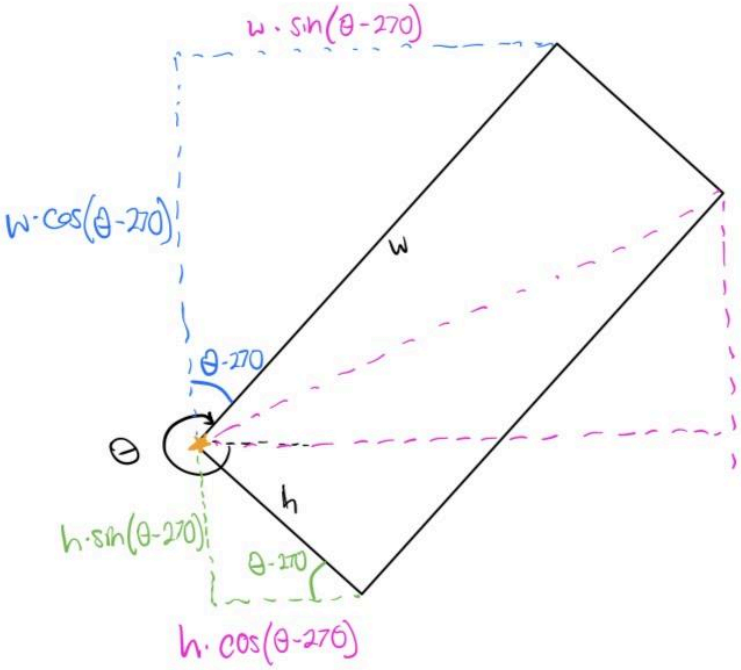
Bottom Bound : 0

Left Bound : $w\cdot\cos(\theta-180^\circ)$

Right Bound : $h\cdot\sin(\theta-180)$

270° → 360°





Top Bound : $w \cdot \cos(\theta - 270)$

Bottom Bound : $h \cdot \sin(\theta - 270)$

Left Bound : 0

Right Bound : $h \cdot \cos(\theta - 270) + w \cdot \sin(\theta - 270)$

When we first implemented our bounding functions they were requiring too much computational power and our draggable objects became jittery. We used trig identities to eliminate arctan from our code and optimized it to reduce how many times sin and cos were being called.

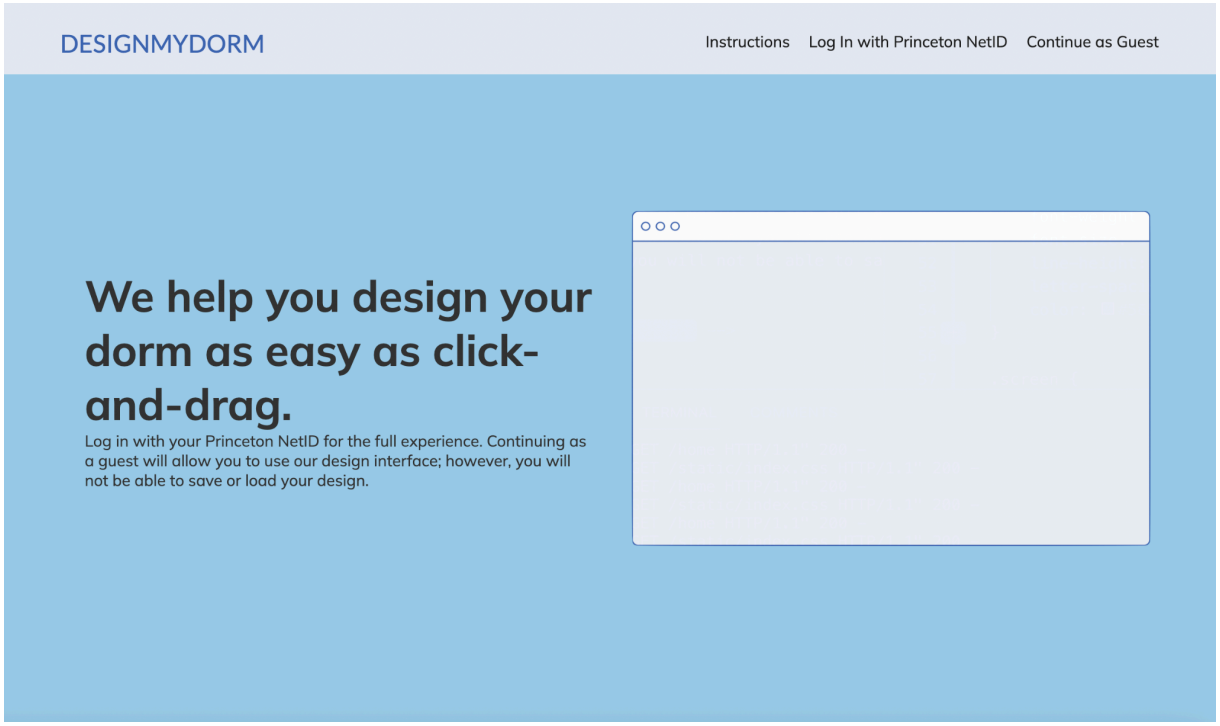
In order to keep a user from stretching a Konva Transformer outside of a stage, we had to add scaling code that reduced the scale of the transformer if it was leaving the stage. This scaling code is fairly complicated, because we needed the shape to only scale down the parts of the transformer that were trying to leave the stage. Otherwise, our boxes would collapse. However, the transformer must collapse in both directions in instances when a corner is hitting a boundary or when the shape is already relatively thin. Thus, we had to add other possible cases for which the transformer is downscaling in both directions. In order to determine what case to do, we have to determine the current size of the transformer and how the user is currently trying to transform it, which involves checking the transformer's current scale and active anchor.

Making Our Website Responsive:

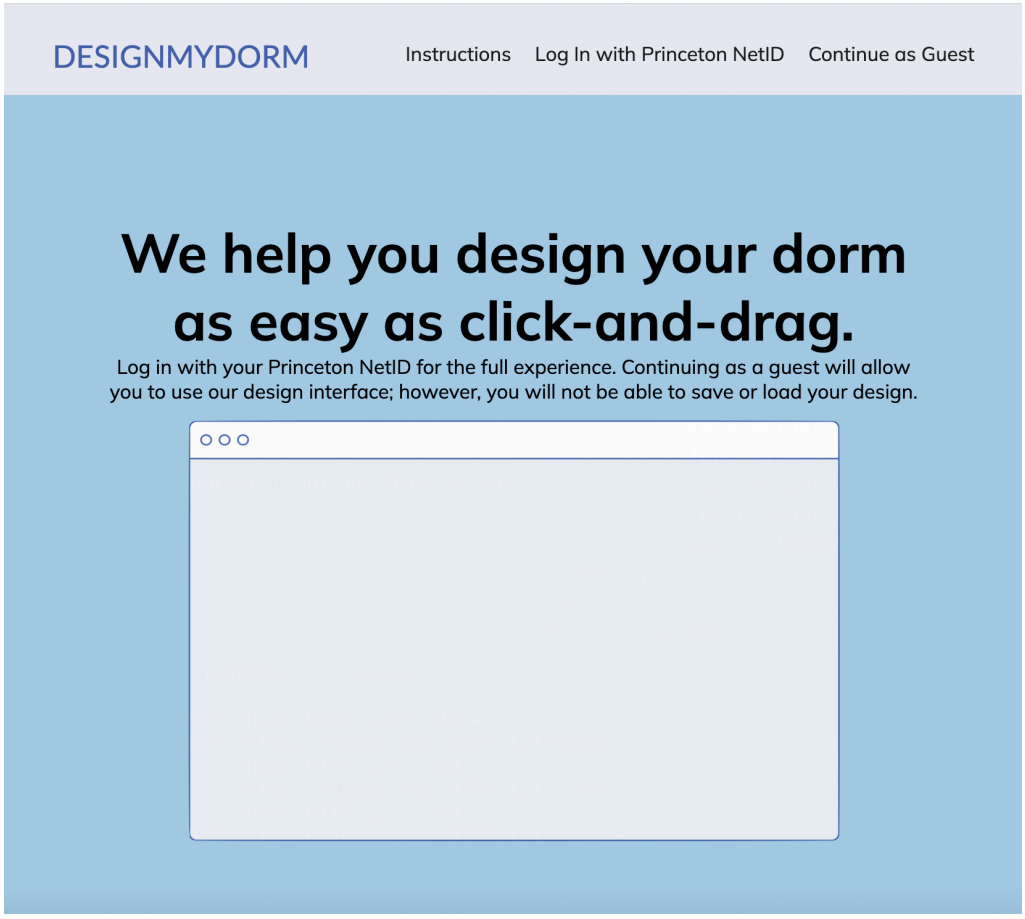
We took several approaches to make our website responsive.

One approach was making a div that contained the layout for a large screen and a div containing the layout for a small screen. We then use media queries to decide which div to display. This method is seen in our landing (also known as Home) page.

Large Screen:

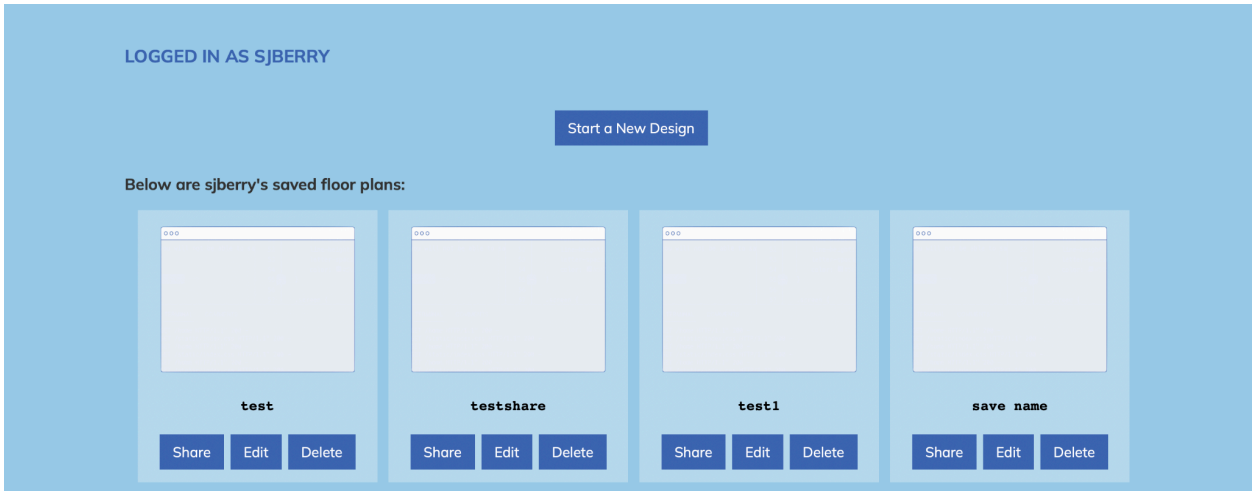


Small Screen:

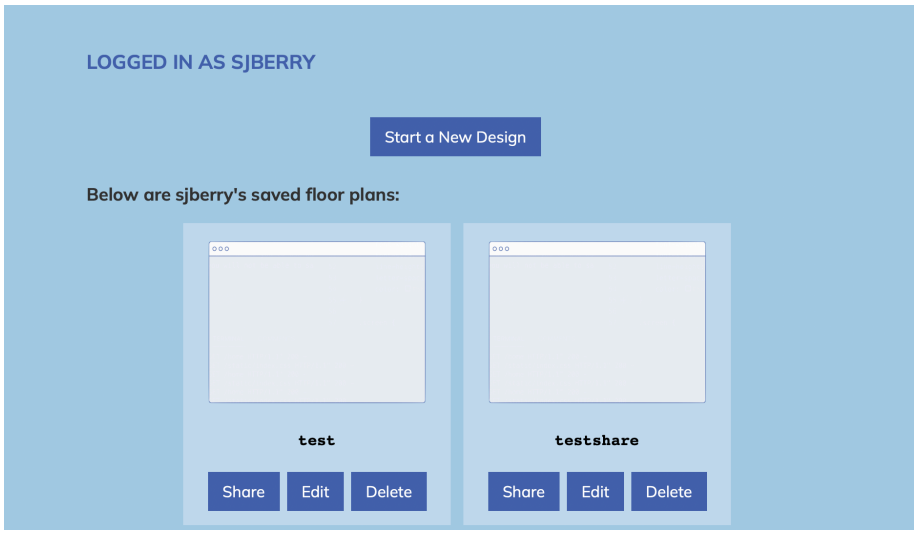


Our second method was putting our information in rectangular divs that were stored in a div with a flex-display. Examples of this include saved design on our Previous Designs' (also known as the Saved Designs') page and the instructions on our scaling page.

Large Screen:

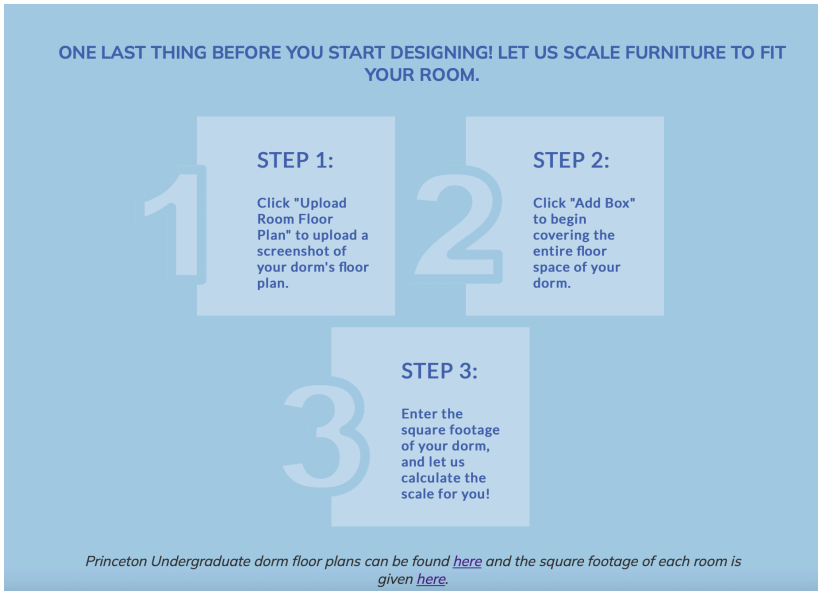


Small Screen:



Large Screen:

Small Screen:



Our third method was to deal with our workspaces. Our workspaces dimensions are based on the width of the user's screen. When the user resizes their screen, we have a function called fitStageIntoContainer that is called by an event listener when the user resizes their screen. This function calculates what the new width and height of the work area should be and resizes the stage accordingly. It then resizes the background image using a canvas to its new size. The function rescales all the furniture or transformer boxes to remain in the same location in the scaled down room. It alters the scale factor, which accounts for the users' window width, so additional furniture added will be at the new scale.

Sending information about which room to share to our JavaScript share function:

Originally, we created a JavaScript that received a room's save name as a parameter and set it as a cookie (as well as the share netid collected by the pop up thrown by the function). However, the save name was contained in a Jinja2 template. When a user included an apostrophe in their save name, the "Share" button would not work since the apostrophe was being interpreted as an unmatched apostrophe. Now when the user clicks a "Share" button for a saved design on the Previous Designs' page, the button's value (a url containing the save name as a key value pair) is passed to a function in load.js. This function uses a pop up to collect the netid of the person the user wants to share the design with. The share netid is then set as a cookie before our program routes to designmydorm.py using the url which was the button's value.

Preventing users from accessing designs that weren't theirs or wasn't shared with them:

We load shared designs by parsing the shared name. Anything before the first underscore is the host's netid and anything after is the original save name. We realize that someone could construct a url to access an unshared design if they knew someone's url and one of their save names. To prevent this from happening, we added a checker to make sure the design was shared with the user by querying our shared_save table to make sure a design was shared with them before loading the design.

Sources

For further references we used to compose this website please consult:
https://docs.google.com/spreadsheets/d/1ydVoHK6sUf5Z_PchedaPeYb63kfR-27wsXA2mcFkLRY/edit#gid=0