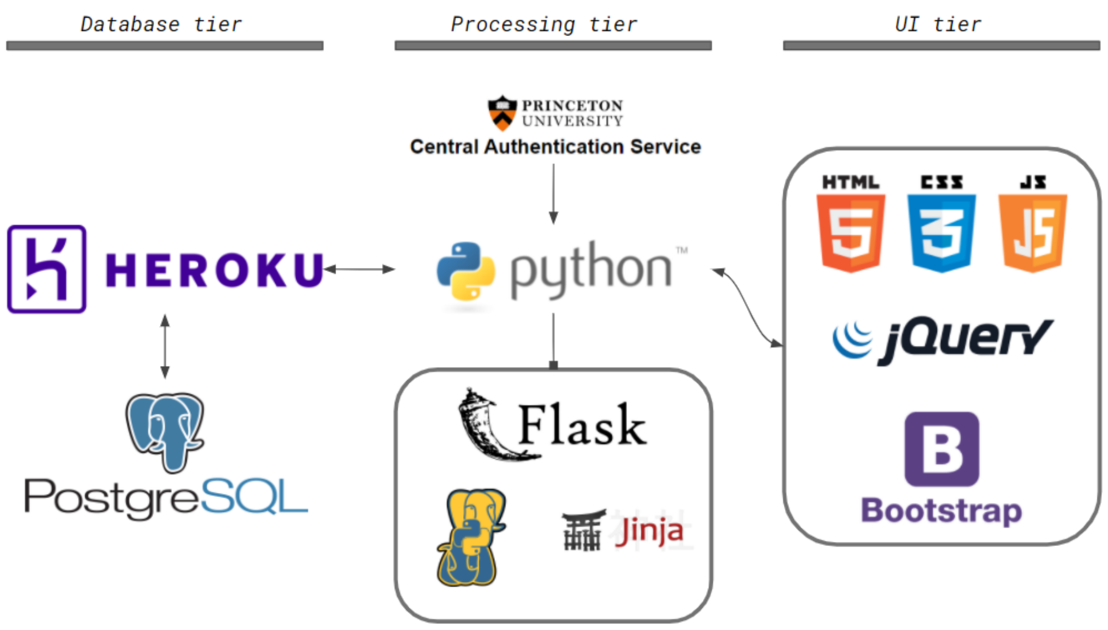
acaprez

Programmer's Guide

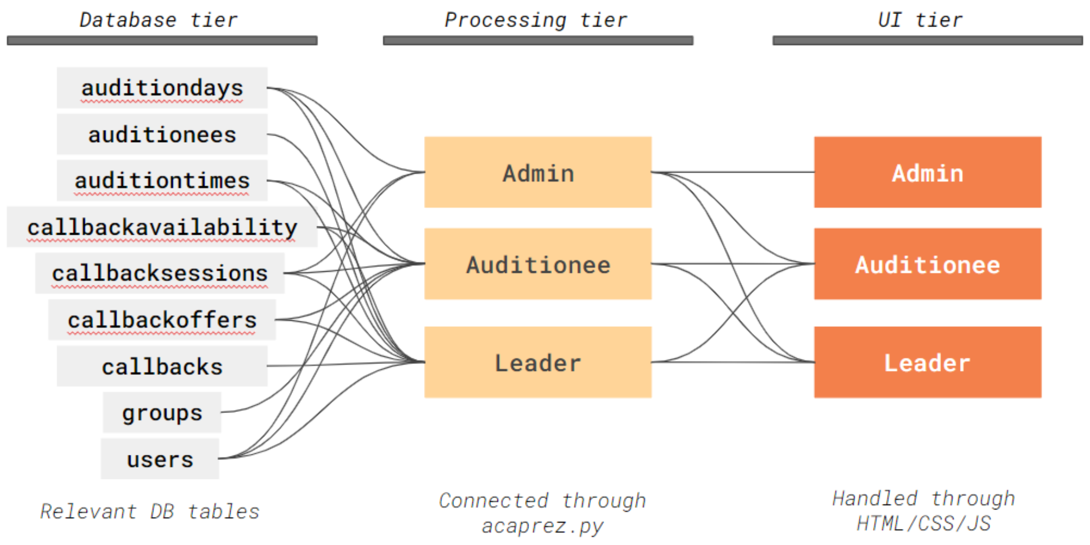
Created By
Tim Manley
Ryan Gibbons
Jane Castleman
Silas Mohr

Application Architecture

Technologies Used:



Top Level Diagram:



In our UI tier, we used a combination of HTML, Jinja Templates, CSS, Bootstrap, vanilla JavaScript, and JQuery/Ajax to render our front end UI. The web pages are rendered using Jinja HTML templates and attached static CSS files. The CSS and Bootstrap style the pages and then the JavaScript/JQuery give the pages responsiveness.

In our processing tier, we used Python, specifically Flask, to handle the requests from users and to connect to the database tier. Flask uses a system of endpoints for each url that the website recognizes and sends an HTTP request to the browser that contains the rendered Jinja template when a user visits a page.

In our database tier, we used a PostgreSQL database hosted through Heroku. The entire application is also hosted through Heroku. To connect to the database, we used the psycopg2 library.

Our three tiers are broken down to handle three use cases: administrators, acapella group leaders, and people signing up for auditions. Each type of user can only view the index/login page and their specific pages, but not those of any other kind of user. The HTTP requests for all three use cases are handled using flask endpoints and are passed through our processing tier. The main templates and endpoints for each use case are as follows:

Users	Endpoints	Templates
Admin	index, login, bypasslogin, netid, about, logoutconfirmation, admin, reset, showgroupcallbacks	insufficient, index, caslogin, about, confirmLogout, admin, availabilityCalendar
Leader	index, login, bypasslogin, netid, about, logoutconfirmation, leader, offercallback, addtimes, addedtimes, canceltime, auditioneeInfo	insufficient, index, caslogin, about, confirmLogout, leader, addtimes, auditioneeInfo
Audition ee	index, login, bypasslogin, netid, about, logoutconfirmation, auditionee, cancelaudition, acceptcallback, editprofile, callbackavailability, addedcallbacks, confirmprofile, showgroupauditions, createAudition, signup-confirmation	insufficient, index, caslogin, about, confirmLogout, editprofile, auditionee, callbackavailability, addedcallbacks, confirmprofile, auditioneeCalendar, createAudition

Processing/UI Tiers

Endpoint	Request Type	Description	Templates
/ or /index	GET	Landing page with links to login and pictures of team members	index
/login	GET	CAS backdoor/developer login page	caslogin
/leader	GET	Dashboard for leader of acapella group with all the groups audition slots and the corresponding auditionee, as well as pending/accepted callbacks	leader, leadernav
/admin	GET	Dashboard for admin with ability to set audition days, callback days/times, see all accepted callbacks and button to reset the website for a new audition cycle	admin, genericnav
/reset	POST	Resets the database for new audition cycle	N/A
/auditionee	GET	Creates new auditionee profile for users who don't have one and a dashboard for auditionees with the auditions they've signed up for and pending/accepted callbacks	editprofile, nav and auditionee, nav
/cancelaudition	POST	Cancels audition for the audtionee user	N/A
/acceptcallback	POST	Accepts callback from given group for the auditionee user	N/A
/offercallback	POST	Offers callback to given user for the leader user	N/A
/editprofile	GET	Allows auditionee user to edit profile information	editprofile, nav

/callbackavailability	GET	Allows auditionee user to set their callback availability	callbackavailability, nav
/addedcallbacks	GET or POST	Adds callback availability for an auditionee user and confirms sucessfully adding the times to the database	addedcallbacks, nav
/addtimes	GET	Allows a leader user to select audition times for their group	addtimes, leadernav, leaderCalendar
/addedtimes	GET or POST	Adds audition times for a leader users group	N/A
/canceltime	POST	Cancels a given audition for a leader user	N/A
/confirmprofile	GET or POST	Confirms profile information after creation or editing	confirmprofile, nav
/netID	GET	Routes user to correct home page	N/A
/bypasslogin	GET	Backdoor for bypassing CAS login which sets permissions based on the provided netid and routes to the correct home page	N/A
/showgroupauditions	GET	Displays all available auditions of a given group for an auditionee user	auditioneeCalendar
/showgroupcallbacks	GET	Displays all accepted callbacks of a given group for an admin user	availabilityCalendar
/createAudition	GET	Displays list of groups and all available auditions, which the user can use to sign up for one, of a selected group for an auditionee user	createAudition, nav
/auditioneeInfo	GET	Displays profile information of auditionee for a leader user	auditioneeInfo, leadernav

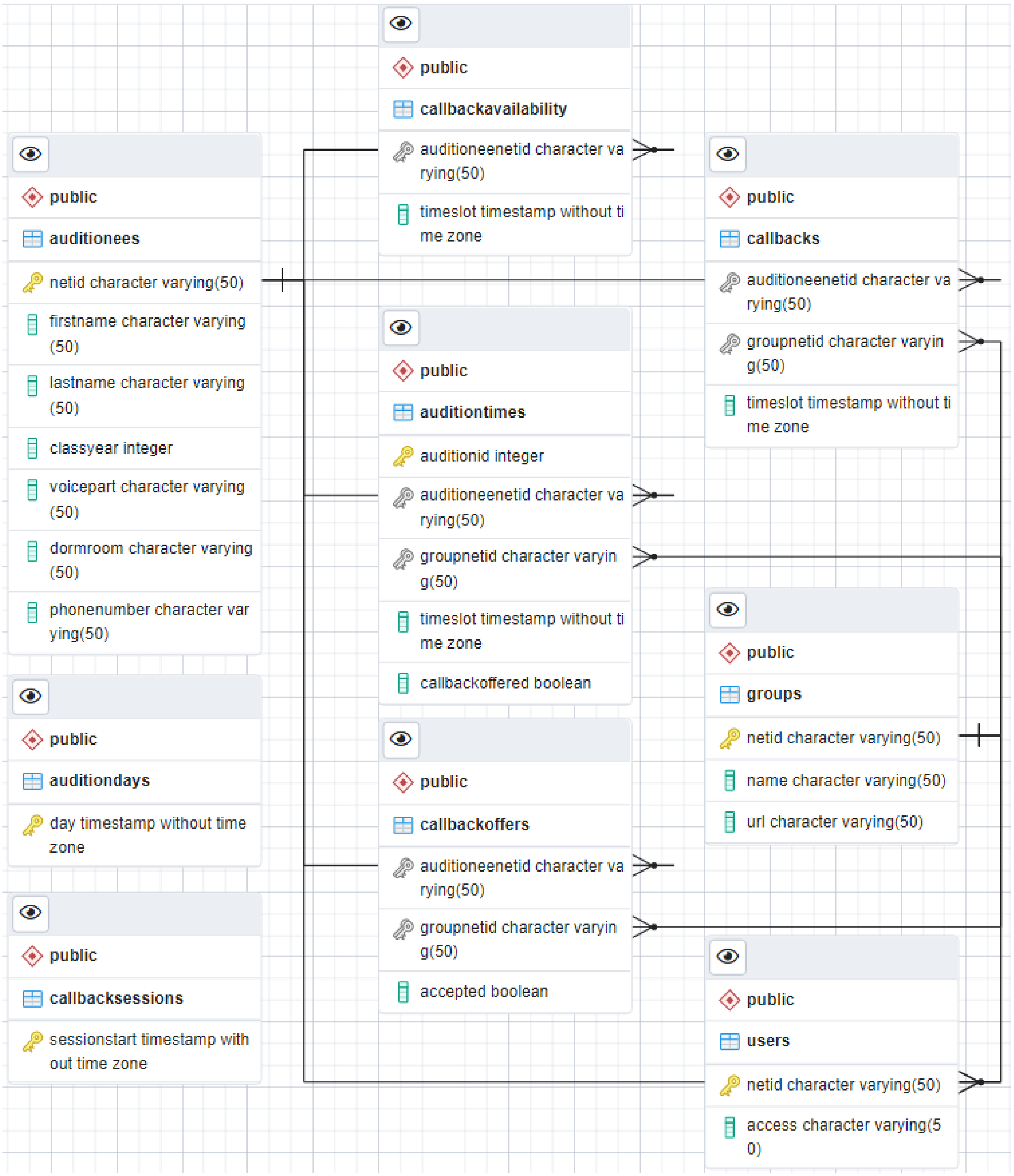
/signup-confirmation	GET or POST	Attempts to sign auditionee up for given audition and returns whether it was successful or not	N/A
/about	GET	Page with information about all groups in Acaprez	about, nav
/logoutconfirmation	GET	Confirms that user wants to log out	confirmlogout, nav

Each endpoint, except for the index page, authenticates the user using the auth.py module and checks permissions before rendering the corresponding template. If the permissions aren't correct, then the user is presented with a page informing them that they don't have the necessary permissions to view the page and the actions that the endpoint would usually perform are bypassed.

All these endpoints are set up and handled within the acaprez.py file and if there is a need to connect to the database, functions from the database.py module are used.

Database Tier

The database is broken down into nine different tables that are initialized by the init_db.py module. The tables are as follows:


public
callbackavailability
auditioneenetid character varying(50)
timeslot timestamp without time zone
public
auditionees
netid character varying(50)
firstname character varying (50)
lastname character varying (50)
classyear integer
voicepart character varying (50)
dormroom character varying (50)
phonenumber character varying(50)
public
callbacks
auditioneenetid character varying(50)
groupnetid character varying(50)
timeslot timestamp without time zone
public
auditiontimes
auditionid integer
auditioneenetid character varying(50)
groupnetid character varying(50)
timeslot timestamp without time zone
callbackoffered boolean
public
groups
netid character varying(50)
name character varying(50)
url character varying(50)
public
auditiondays
day timestamp without time zone
public
callbackoffers
auditioneenetid character varying(50)
groupnetid character varying(50)
accepted boolean
public
callbacksessions
sessionstart timestamp without time zone
public
users
netid character varying(50)
access character varying(50)

When the database is reset for a new audition cycle, all the tables are cleared and then some of them are populated with certain default values. The ‘groups’ table will be populated with the existing Acaprez groups’ names, netids and urls. The ‘users’ table will include the netids of each member of Acaprez and the netid of the current administrator. To change these netids or add more groups if they become part of Acaprez, the administrator must go into init_db.reset_database() and add either a new call to add_group(cur, <netid>, <Name>, <website>') or change the admins netid to the new netid in the code.

The ‘auditiondays’ and ‘callbacksessions’ tables will be populated with the days/times set by the administrator during the reset. The rest of the tables will be empty until they are populated by users. The database module is heavily documented with docstrings for each function and the datatype it is expected to return.

When a leader adds audition times for their group, they get put into the ‘auditiontimes’ table. The ‘callbacks’ and ‘callbackoffers’ tables are also populated by the leader.

The ‘auditionees’ table is populated when a new user logs in and creates a profile. These users can then sign up for auditions and their netid is updated in the ‘auditiontimes’ table. The ‘callbackavailability’ table is similarly populated by an auditionee.

Design Problems & Solutions

Using Bootstrap

- At first, it was tedious to get comfortable with the Bootstrap grid layout, specifically when applying styling. Sometimes it would take trial and error to ensure that styling was applied to the right layer of HTML code (i.e. to the right <div> element), targeting the desired row, column, or specific element. We solved this problem by closely reading Bootstrap documentation, reviewing lecture code, and a lot of trial and error to see when and how styling would appear.

Callback Availability Scheduling

- When designing our callbacks system, we originally had the difficulty of trying to determine how to assign/select callback availability. The idea started with a sort of "flipped" calendar design to our existing audition sign-up system, with auditionees painting over times they were available for and leaders selecting one time for them to return. However, the callbacks system differs from the original audition sign-up in that there are typically only four or five total time slots in which callbacks are held, which are shared among the 8 groups. This led us to a system in which the admin would determine times, with auditionees selecting checkboxes for which they were available rather than painting over a range of times. This also solved the problem of callbacks not necessarily spanning the same range of time as our 15-minute increment audition slots.

Optimal Time Slot Input Forms

- After settling on the callback system described above, we had to determine the neatest way for the admin to enter these callback times for an upcoming audition season. This critically differed from the scheduling of the first round of auditions, in which the admin simply selects several days from a calendar which are automatically populated with a range of times for groups to then select from. None of the JavaScript calendar packages we could find adequately met the need of being able to schedule multiple dates along with unique times, leading us to the solution of adding a button that would use JQuery to add more form fields each designating a unique date-time combination.