acaprez

Product Evaluation

Created By

Tim Manley

Ryan Gibbons

Jane Castleman

Silas Mohr

Table of Contents

Testing

In general, we tried to use automation as much as possible whilst testing. Specifically, we used the built-in unittest module to test aspects of the *database.py* module, and pytest in conjunction with the coverage tool to automate testing the application as a whole. That said, there was a lot of non-automated testing (particularly for UI related issues), which was ongoing throughout the project, and has been continued by our user testers to remove bias (See Evaluation By Users and Appendix B for more information about our user testing).

Internal Testing

For our internal testing, we focussed primarily on our *database.py* module. This is because it is by far the largest module, as well as the most complex, and it is also the core of our project - if it fails, so does the whole project. Another reason to focus on this module is that it is where the majority of errors could be thrown, since it is the lowest level of our code hierarchy and interacts directly with the PostgreSQL database hosted on Heroku.

Function/Method Failures

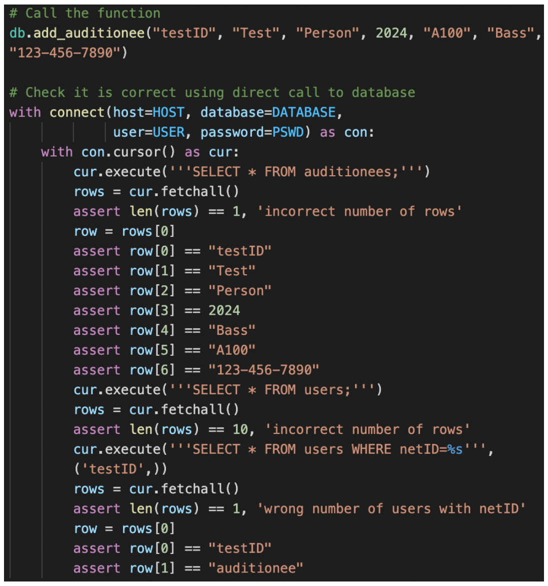
database.py

The *database.py* file is tested with the file named: *testdb.py*. We made sure to run *testdb.py* before every deployment. Our method of testing each function for function failures is as follows:

1. Call the function with valid parameters.
2. If the function modifies the database and does not return anything:
    a. Execute an SQL query to assert that the modification is as expected.
3. If the function does not modify the database and does return something:
    a. Check that what is returned is what is expected.

Since every function either modifies the database and returns nothing, or doesn't modify the database and returns something, we have thus covered every kind of function. It also makes sense to test those functions which modify the database first, since then we can use them to ensure the other functions work correctly.

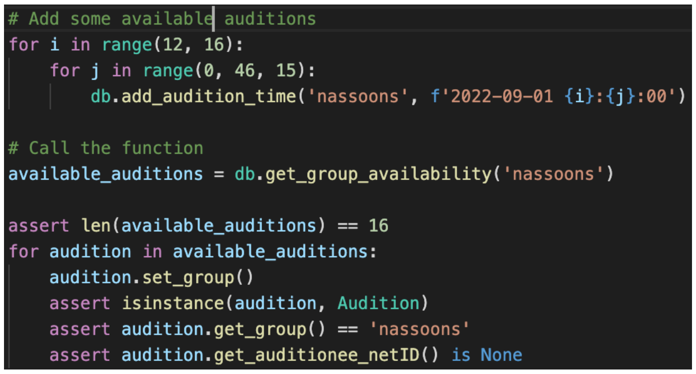
An example of testing a modifying function (*add_auditionee* in this case):



```
# Call the function
db.add_auditionee("testID", "Test", "Person", 2024, "A100", "Bass",
"123-456-7890")

# Check it is correct using direct call to database
with connect(host=HOST, database=DATABASE,
             user=USER, password=PSWD) as con:
    with con.cursor() as cur:
        cur.execute('''SELECT * FROM auditionees;''')
        rows = cur.fetchall()
        assert len(rows) == 1, 'incorrect number of rows'
        row = rows[0]
        assert row[0] == "testID"
        assert row[1] == "Test"
        assert row[2] == "Person"
        assert row[3] == 2024
        assert row[4] == "Bass"
        assert row[5] == "A100"
        assert row[6] == "123-456-7890"
        cur.execute('''SELECT * FROM users;''')
        rows = cur.fetchall()
        assert len(rows) == 10, 'incorrect number of rows'
        cur.execute('''SELECT * FROM users WHERE netID=%s''',
        ('testID',))
        rows = cur.fetchall()
        assert len(rows) == 1, 'wrong number of users with netID'
        row = rows[0]
        assert row[0] == "testID"
        assert row[1] == "auditionee"
```

As you can see above, we first call the function, and then we connect to the database and execute two distinct queries to ensure that the relevant tables have been updated correctly.

An example of testing a non modifying function (*get_group_availability* in this case):



```
# Add some available auditions
for i in range(12, 16):
    for j in range(0, 46, 15):
        db.add_audition_time('nassoons', f'2022-09-01 {i}:{j}:00')

# Call the function
available_auditions = db.get_group_availability('nassoons')

assert len(available_auditions) == 16
for audition in available_auditions:
    audition.set_group()
    assert isinstance(audition, Audition)
    assert audition.get_group() == 'nassoons'
    assert audition.get_auditionee_netID() is None
```

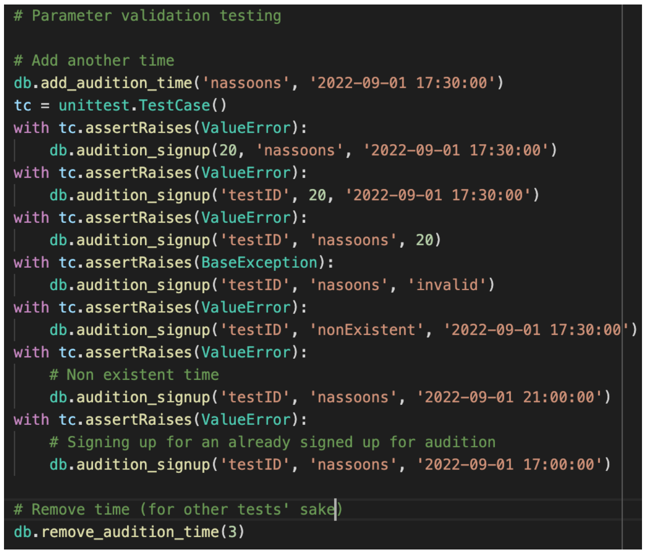
Once again, as you see above, we first use one of the previously tested functions (*add_audition_time*) to populate the database. We then call *get_group_availability* and assert that what is returned is correct.

Validate Parameters

database.py

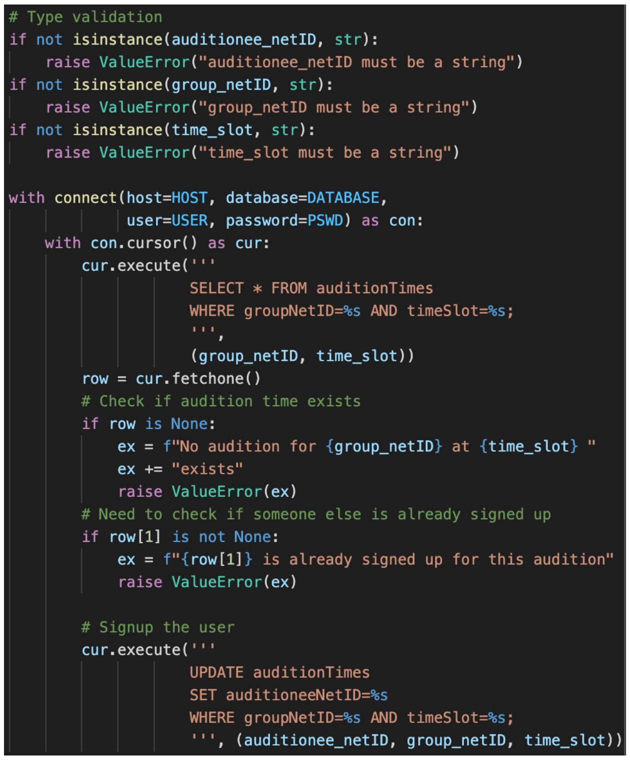
Once again, *database.py* is tested by *testdb.py*. Our method for testing parameter validation is quite simple. Within a *test_{method name}* function, we use Python's *unittest* library, specifically the *unittest.TestCase().assertRaises(ValueError)* function, to check that calling a function with invalid parameters raises a *ValueError*.

An example of this (inside *test_audition_signup*):



```python
# Parameter validation testing

# Add another time
db.add_audition_time('nassoons', '2022-09-01 17:30:00')
tc = unittest.TestCase()
with tc.assertRaises(ValueError):
    db.audition_signup(20, 'nassoons', '2022-09-01 17:30:00')
with tc.assertRaises(ValueError):
    db.audition_signup('testID', 20, '2022-09-01 17:30:00')
with tc.assertRaises(ValueError):
    db.audition_signup('testID', 'nassoons', 20)
with tc.assertRaises(BaseException):
    db.audition_signup('testID', 'nasoons', 'invalid')
with tc.assertRaises(ValueError):
    db.audition_signup('testID', 'nonExistent', '2022-09-01 17:30:00')
with tc.assertRaises(ValueError):
    # Non existent time
    db.audition_signup('testID', 'nassoons', '2022-09-01 21:00:00')
with tc.assertRaises(ValueError):
    # Signing up for an already signed up for audition
    db.audition_signup('testID', 'nassoons', '2022-09-01 17:00:00')

# Remove time (for other tests' sake)
db.remove_audition_time(3)
```

And the associated function with input validation in the database module:

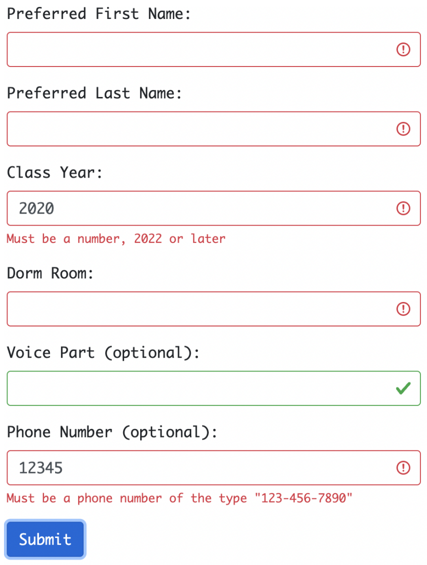


```python
# Type validation
if not isinstance(auditionee_netID, str):
    raise ValueError("auditionee_netID must be a string")
if not isinstance(group_netID, str):
    raise ValueError("group_netID must be a string")
if not isinstance(time_slot, str):
    raise ValueError("time_slot must be a string")

with connect(host=HOST, database=DATABASE,
             user=USER, password=PSWD) as con:
    with con.cursor() as cur:
        cur.execute('''
                    SELECT * FROM auditionTimes
                    WHERE groupNetID=%s AND timeSlot=%s;
                    ''',
                    (group_netID, time_slot))
        row = cur.fetchone()
        # Check if audition time exists
        if row is None:
            ex = f"No audition for {group_netID} at {time_slot} "
            ex += "exists"
            raise ValueError(ex)
        # Need to check if someone else is already signed up
        if row[1] is not None:
            ex = f"{row[1]} is already signed up for this audition"
            raise ValueError(ex)

        # Signup the user
        cur.execute('''
                    UPDATE auditionTimes
                    SET auditioneeNetID=%s
                    WHERE groupNetID=%s AND timeSlot=%s;
                    ''', (auditionee_netID, group_netID, time_slot))
```

editprofile.html

Another part of our code needing parameter validation is *editprofile.html*, since this is where we allow the user to modify their profile, using text fields in a form submission. It is these text fields which need input validation. We achieved this using the built in HTML input validation, and tested the form rigorously during development.

Screenshot of form with invalid inputs, demonstrating HTML input validation:



Preferred First Name:

Preferred Last Name:

Class Year:

2020

Must be a number, 2022 or later

Dorm Room:

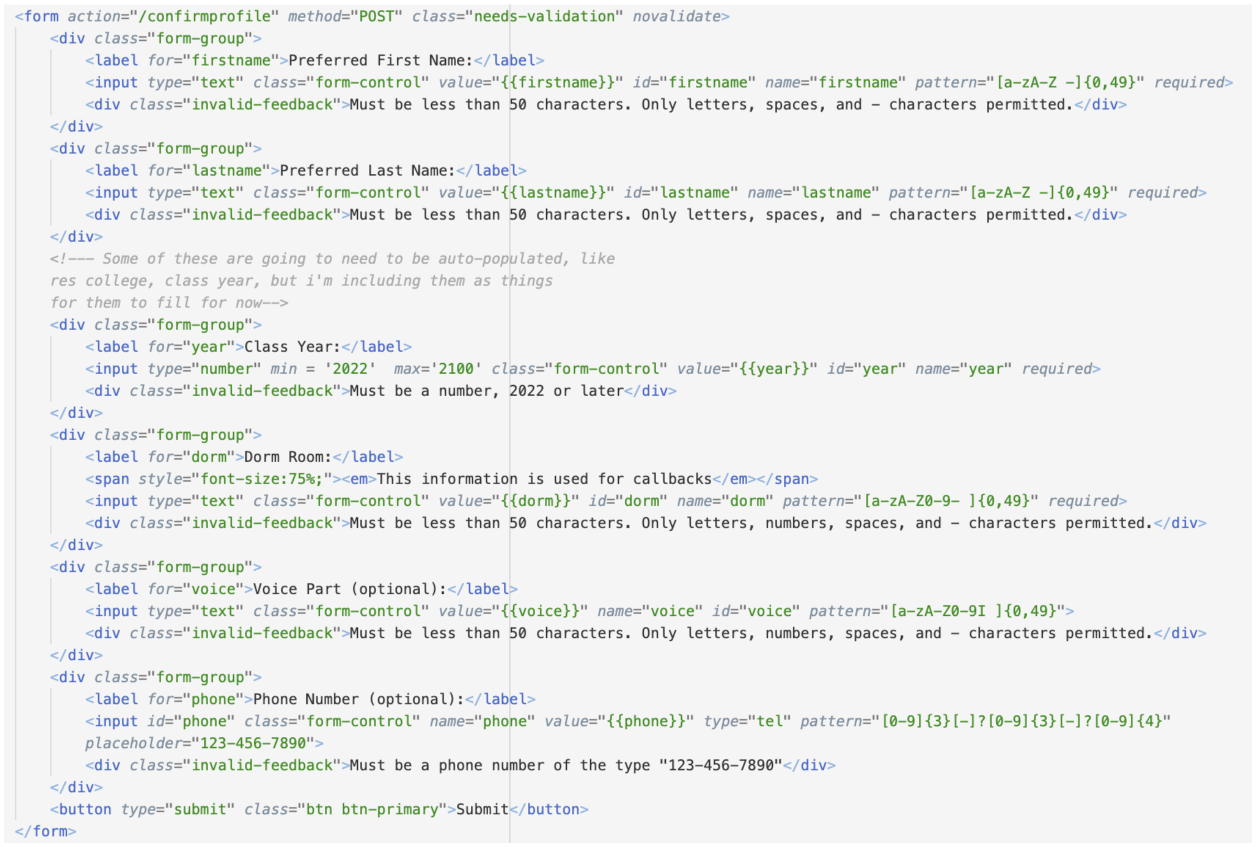
Voice Part (optional):

Phone Number (optional):

12345

Must be a phone number of the type "123-456-7890"

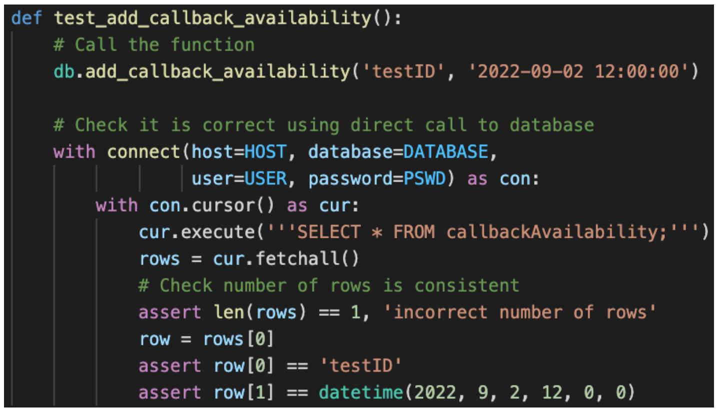
HTML code where we validate the inputs:



```
<form action="/confirmprofile" method="POST" class="needs-validation" novalidate>
    <div class="form-group">
        <label for="firstname">Preferred First Name:</label>
        <input type="text" class="form-control" value="{{firstname}}" id="firstname" name="firstname" pattern="[a-zA-Z -]{0,49}" required>
        <div class="invalid-feedback">Must be less than 50 characters. Only letters, spaces, and - characters permitted.</div>
    </div>
    <div class="form-group">
        <label for="lastname">Preferred Last Name:</label>
        <input type="text" class="form-control" value="{{lastname}}" id="lastname" name="lastname" pattern="[a-zA-Z -]{0,49}" required>
        <div class="invalid-feedback">Must be less than 50 characters. Only letters, spaces, and - characters permitted.</div>
    </div>
    <!--- Some of these are going to need to be auto-populated, like
    res college, class year, but i'm including them as things
    for them to fill for now-->
    <div class="form-group">
        <label for="year">Class Year:</label>
        <input type="number" min = '2022'  max='2100' class="form-control" value="{{year}}" id="year" name="year" required>
        <div class="invalid-feedback">Must be a number, 2022 or later</div>
    </div>
    <div class="form-group">
        <label for="dorm">Dorm Room:</label>
        <span style="font-size:75%;"><em>This information is used for callbacks</em></span>
        <input type="text" class="form-control" value="{{dorm}}" id="dorm" name="dorm" pattern="[a-zA-Z0-9- ]{0,49}" required>
        <div class="invalid-feedback">Must be less than 50 characters. Only letters, numbers, spaces, and - characters permitted.</div>
    </div>
    <div class="form-group">
        <label for="voice">Voice Part (optional):</label>
        <input type="text" class="form-control" value="{{voice}}" name="voice" id="voice" pattern="[a-zA-Z0-9I ]{0,49}">
        <div class="invalid-feedback">Must be less than 50 characters. Only letters, numbers, spaces, and - characters permitted.</div>
    </div>
    <div class="form-group">
        <label for="phone">Phone Number (optional):</label>
        <input id="phone" class="form-control" name="phone" value="{{phone}}" type="tel" pattern="[0-9]{3}[-]?[0-9]{3}[-]?[0-9]{4}"
        placeholder="123-456-7890">
        <div class="invalid-feedback">Must be a phone number of the type "123-456-7890"</div>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Check Invariants

We don't really maintain any persistent data structures within our code, since all the data is hosted in our PostgreSQL database on Heroku. We do however ensure that the number of rows in each table is maintained after database function calls. This is tested in most functions within *testdb.py*, an example of this is shown below.



```python
def test_add_callback_availability():
    # Call the function
    db.add_callback_availability('testID', '2022-09-02 12:00:00')

    # Check it is correct using direct call to database
    with connect(host=HOST, database=DATABASE,
                 user=USER, password=PSWD) as con:
        with con.cursor() as cur:
            cur.execute('''SELECT * FROM callbackAvailability;''')
            rows = cur.fetchall()
            # Check number of rows is consistent
            assert len(rows) == 1, 'incorrect number of rows'
            row = rows[0]
            assert row[0] == 'testID'
            assert row[1] == datetime(2022, 9, 2, 12, 0, 0)
```

External Testing

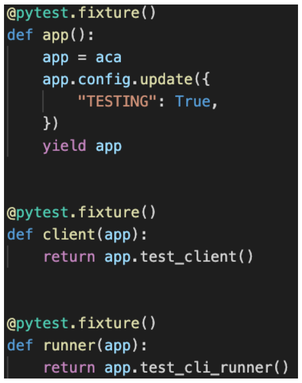
Our external testing was focused on the application as a whole, using testing tools such as pytest and coverage to evaluate the application. Our method essentially boiled down to automating the process of being a user by sending specific requests to the server, and then evaluating aspects of the response, such as HTTP status codes and raw HTML. It is hard to be 100% rigorous that the response is fully correct, since the only way to do so would be to compare the response to an entire HTML file, which leaves little room for modification to the application, so we decided to look for certain indicators in the response that would most clearly convey that the response is as we expect. For example, after signing up for an audition we need only check that the audition appears somewhere in both the auditionee's dashboard HTML and the leader's dashboard HTML, since the rest of the HTML on both pages remains largely unchanged, whether the response was correct or not.

White Box

Statement Testing

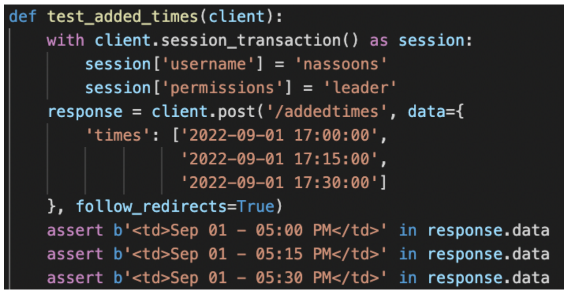
For statement testing, we used pytest in conjunction with the coverage tool. In order to navigate the web app, we used the flask *app.test_client()*. The file *statement_test.py* has all of the test functions. (Source: https://flask.palletsprojects.com/en/2.1.x/testing/)

This is the pytest set-up, where 'aca' is the Flask app created in *acaprez.py*:



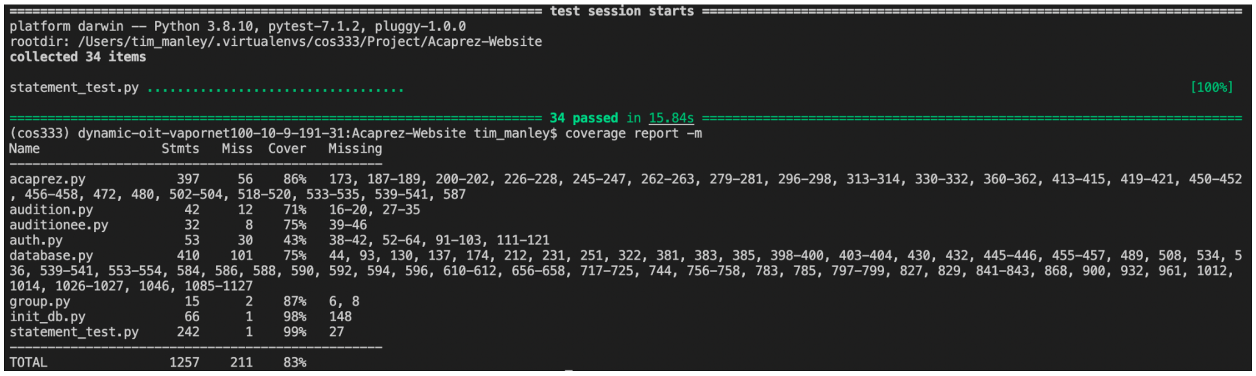
```python
@pytest.fixture()
def app():
    app = aca
    app.config.update({
        "TESTING": True,
    })
    yield app


@pytest.fixture()
def client(app):
    return app.test_client()


@pytest.fixture()
def runner(app):
    return app.test_cli_runner()
```

An example of a statement test function used in *statement_test.py*:



```python
def test_added_times(client):
    with client.session_transaction() as session:
        session['username'] = 'nassoons'
        session['permissions'] = 'leader'
    response = client.post('/addedtimes', data={
        'times': ['2022-09-01 17:00:00',
                  '2022-09-01 17:15:00',
                  '2022-09-01 17:30:00']
    }, follow_redirects=True)
    assert b'<td>Sep 01 - 05:00 PM</td>' in response.data
    assert b'<td>Sep 01 - 05:15 PM</td>' in response.data
    assert b'<td>Sep 01 - 05:30 PM</td>' in response.data
```

The coverage report generated by the coverage tool:



```
===================================================================== test session starts ======================================================================
platform darwin -- Python 3.8.10, pytest-7.1.2, pluggy-1.0.0
rootdir: /Users/tim_manley/.virtualenvs/cos333/Project/Acaprez-Website
collected 34 items

statement_test.py ..................................                                                                                                     [100%]

====================================================================== 34 passed in 15.84s ======================================================================
(cos333) dynamic-oit-vapornet100-10-9-191-31:Acaprez-Website tim_manley$ coverage report -m
Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
acaprez.py            397     56    86%   173, 187-189, 200-202, 226-228, 245-247, 262-263, 279-281, 296-298, 313-314, 330-332, 360-362, 413-415, 419-421, 450-452
, 456-458, 472, 480, 502-504, 518-520, 533-535, 539-541, 587
audition.py            42     12    71%   16-20, 27-35
auditionee.py          32      8    75%   39-46
auth.py                53     30    43%   38-42, 52-64, 91-103, 111-121
database.py           410    101    75%   44, 93, 130, 137, 174, 212, 231, 251, 322, 381, 383, 385, 398-400, 403-404, 430, 432, 445-446, 455-457, 489, 508, 534, 5
36, 539-541, 553-554, 584, 586, 588, 590, 592, 594, 596, 610-612, 656-658, 717-725, 744, 756-758, 783, 785, 797-799, 827, 829, 841-843, 868, 900, 932, 961, 1012,
1014, 1026-1027, 1046, 1085-1127
group.py               15      2    87%   6, 8
init_db.py             66      1    98%   148
statement_test.py     242      1    99%   27
-------------------------------------------------
TOTAL                1257    211    83%
```

As can be seen above, we covered 83% of statements in our program, with no failed tests. The reason we were unable to achieve higher is because of a few things. Within *acaprez.py* there are a number of edge cases we look for which are impossible to achieve within the constraints of pytest and the flask test client, which meant we couldn't test them. A few of the methods in *audition.py*, *auditionee.py*, and *group.py* are there as getters and setters, and they are not currently called by our program, so we can't access them, but they are there to future proof our code. The *auth.py* module was hard to test because pytest doesn't allow external redirects, so a lot of the paths were impossible to follow. Finally, *database.py* has a number of functions that are also there for future use, but cannot be called currently, and similarly to *acaprez.py*, there are a number of cases (particularly for error handling) which are impossible to recreate through pytest and the flask test client.

Path Testing

Our statement testing actually covered most logical paths too, since the only way to perform certain requests is by first performing other requests (e.g. can't sign up for an audition without a leader creating an audition time, which can't be done until the admin sets the dates). We also conducted non-automated path testing ourselves throughout development, and after finishing development. One bug that this non-automated testing unveiled was that if you submit changes after editing your profile, you are taken to a confirm page giving you the options to 'confirm' or 'go back'. If instead of confirming you press go back, and then return to the auditionee dashboard, the changes have been saved rather than undoing as should be the case. Since this is a relatively inconsequential bug that was discovered after our development it is still in the program.

Boundary Testing

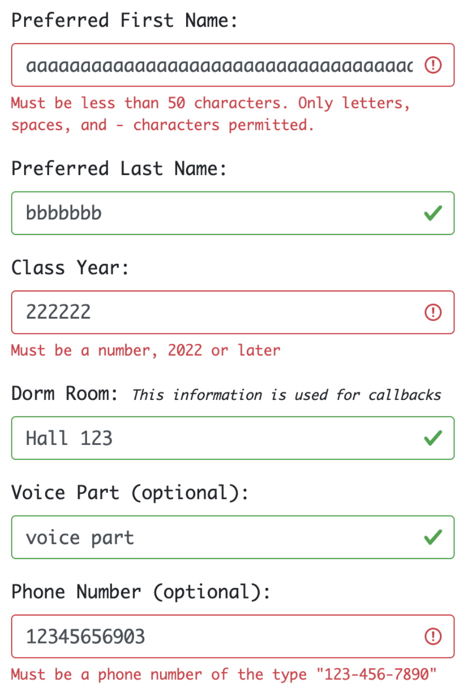
Since there aren't many opportunities to input data, there wasn't a whole boundary testing to do. However, in the areas where we can input data, we did. There were three areas to consider: the input form for profile editing, the audition time scheduler for the leader, and the date input for the admin:

Profile Editor

The main way of doing boundary testing here is to input very long strings for the fields. Our database uses the varchar(50) datatype to store the string fields, thus any strings over 50 characters should be rejected. Testing each field 1-by-1:

- First name: Putting over 50 characters initially yielded an internal server error. This is because we never validated the input length, so the program accepts it and tries to put a record into the database, which then throws an error. We then fixed this by adding a regular expression in the "pattern" parameter controlling input length, and on retesting inputs over 50 characters were rejected. Typing exactly 50 characters works as intended though, albeit with strange formatting on the auditionee dashboard.
- Last name: Same as above.
- Class year: The integer limit as defined by PostgreSQL is 2147483647, so inputting this should work, whereas any values higher than this should be rejected. First, we tested putting this number in, which worked correctly as expected. Now putting in 2147483648 in instead yielded a server error, for the same reason as with first and last name initially. We fixed this by adding a regular expression in the "pattern" parameter controlling integer size, and on retesting it works as desired.
- Dorm room and voice part: These are both varchar(50) data types, and as such we had the same problem needing the same fix as with first name and last name.
- Phone number: This is also a varchar(50) data type, however since we already included precise string formatting validation limiting the number of characters, we don't run into the same problem as with the others, and it works just as intended. However, based on user difficulty with adding hyphens, we changed this to make hyphens optional by slightly modifying our regular expression.

Below is an example of how the form validation now looks, after performing our boundary testing.


Preferred First Name:

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaac

Must be less than 50 characters. Only letters, spaces, and - characters permitted.

Preferred Last Name:

bbbbbbb ✓

Class Year:

222222

Must be a number, 2022 or later

Dorm Room: *This information is used for callbacks*

Hall 123 ✓

Voice Part (optional):

voice part ✓

Phone Number (optional):

12345656903

Must be a phone number of the type "123-456-7890"

Audition time scheduler

The only way to really test the ‘boundary’ of this input is to select every possible time on a day, so we did just that. This worked fine, as expected since each time corresponds to a row in the database, which should have enough space to deal with every time.

Audition/callback date pickers

To test the boundaries of this, we first tested with no dates. This resets the website correctly, and means that the leaders can’t select any times. Although we didn’t strictly specify this behavior, it makes sense and is likely what we would have decided to do anyway, since it wouldn’t make sense to be able to choose times on a non-existent date.
Next we tested by inputting lots of dates for auditions (7 in this case, since the admin is assumed to have some kind of training and auditions never last more than a week). This works as intended, giving the leader more dates to schedule auditions, and also showing more dates and available times on the auditionee side.
Finally, we tested by inputting lots of dates for callbacks (7 again for the same reason as above). This worked fine for the leader side (since nothing is impacted), as well as for the auditionee side (just gave more options when stating callback availability). It also worked ‘as intended’ for the admin side, however the formatting was a little strange with no space between the ‘Available’ labels in the table. Typically, however, there are 3-5 callback sessions, so this shouldn’t be an issue, and even if there were 7 callback sessions one year, it is still readable and understandable, which is the most important thing.

Black Box

Since we are all involved in the project, and none of us happens to know a QA engineer, a lot of black box testing is impossible for us. However, we can estimate how our product will cope under high loads based on Heroku's pricing tier. There can be up to 10,000 rows in our database, so assuming all 8 groups fill up 4 days of audition slots (20 slots per day), that equals 640 rows. If we then include an upper limit on auditionees of 1,500 (approximate class size at Princeton) then with everyone being offered a callback (+1,500) and everyone scheduling for 4 sessions of callbacks (+6000), we get a total number of ~9640 rows. So in the extreme case (which is not going to happen with relative certainty) we get close to, but don't go over Heroku's free tier limit. The only limitation that we could potentially hit is the upper limit of 20 simultaneous connections, however we don't keep persistent database connections by design, so in the worst case someone would just have to wait longer for their request to go through.

Field

We haven't been able to do a full-scale field test, however we hope to schedule some "mock auditions" over the summer for people to participate in and see how our system copes. The main reason we haven't been able to do a full-scale field test is because it requires the participation of many students over a number of days, which during semester time is near impossible to achieve. We do however hope that people will be willing and able to participate over the summer.

Evaluation By Users

Note: the complete task list can be found in Appendix A and user interview notes can be found in Appendix B.

Positive feedback we received

The majority of our users responded positively when asked if the site felt intuitive, and many commented that the task list, even without descriptions, was easy to accomplish. They also said that even if we had not given them a task list, they would have been able to understand how to use the site to sign up for auditions, offer auditions, and act as the site admin.

More specifically, we received positive feedback on our use of concise but informative buttons to navigate between site pages, as well as the consistency of having a navigation bar.

Overall, the users who had previous experience with the acapella audition system greatly preferred this site to the old system. One of the biggest improvements was the centralized nature of the platform, and the ability to sign up for auditions from different groups on the same site. Another improvement is the impossibility of signing up for conflicting auditions or accepting more than two callbacks accidentally. Our site ensures that these errors are impossible to make.

Negative feedback we received

There were a handful of areas when users struggled to understand the optimal way to interact with our user interface.

The most notable point of difficulty was on the admin side, specifically selecting audition dates versus selecting callback timeslots. Audition dates can be selected simultaneously with one calendar selection, while callback timeslots must be selected individually. Users were confused why they were not able to select multiple callback timeslots with the same calendar input. To fix this problem, we changed the “Add Row” button to an “Add timeslot” button to be more informative, and added a note that explained that callback timeslots must be added individually.

Another point of difficulty was the “Cancel” button that was intended to serve as an exit from the “Sign up for an audition” page to return to the auditionee landing. Users did not realize to use this button, and would instead ask for help or use the navigation bar. This button’s label was confusing because it implies canceling the audition(s) a user is signing up for, which is inaccurate. To alleviate this confusion, we changed the button’s label from “Cancel” to “Return to Profile”.

Some of our users were confused about the purpose of some information, like the voice part and dorm room fields on the user profile form. These users were not a part of the Princeton Acapella community and did not intend on auditioning in the future. Although we would assume users would have some knowledge about the audition process, we did add a note on our form that explained that dorm room was necessary information for callbacks, and made the voice part field optional if people were unfamiliar with this term.

Summary

Overall, our users mostly felt as though the site was intuitive to navigate, even upon their first visit. Still, conducting user interviews was important in streamlining our user interface and eliminating any small areas of confusion.

Based on the user feedback, we were able to improve our user interface by removing points of confusion. In interviews after these changes, we noticed that the interviewees were less confused, and didn't struggle with the tasks as much. For example, something as small as changing the "Cancel" button on auditionee sign up to "Return to Profile" made the site easier to navigate.

In the future, we plan to conduct more rigorous user testing with acapella group members and group leaders to prepare the site for the auditions in the fall semester.

Evaluation By Experts

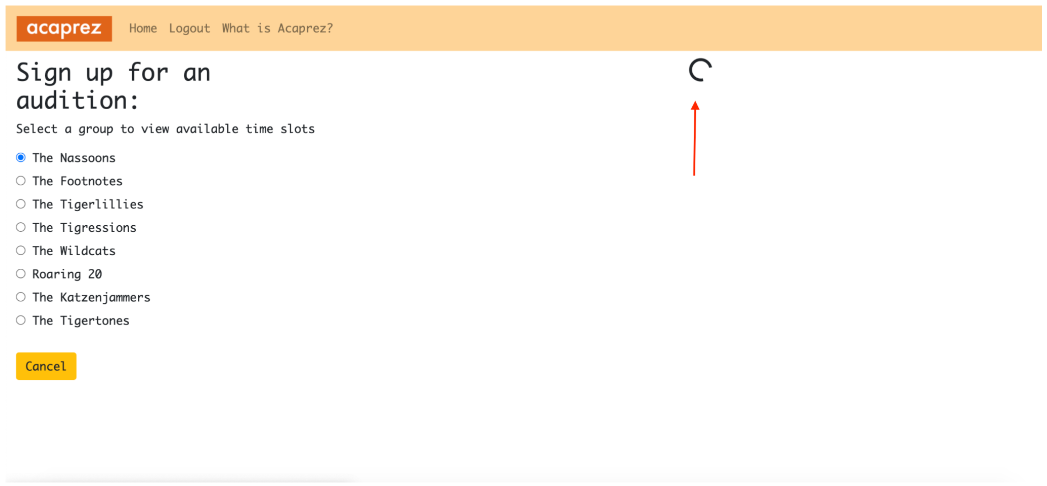
For our evaluation by experts, we used the 10 heuristics by Jakob Nielsen as discussed in lecture.

1. Visibility of system statuses

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time

We use a number of messages, and visuals to help users understand how our system has responded to their actions.

- Loading icon for audition times

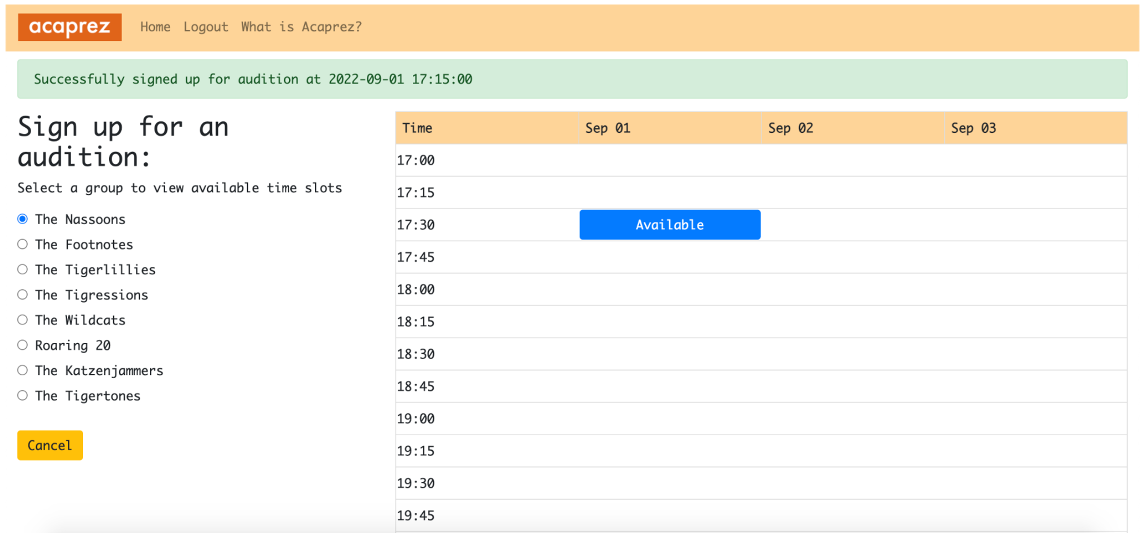


Sign up for an audition:

Select a group to view available time slots

- ● The Nassoons
- ○ The Footnotes
- ○ The Tigerlillies
- ○ The Tigressions
- ○ The Wildcats
- ○ Roaring 20
- ○ The Katzenjammers
- ○ The Tigertones

- Confirmation banner for signing up for an audition



Successfully signed up for audition at 2022-09-01 17:15:00

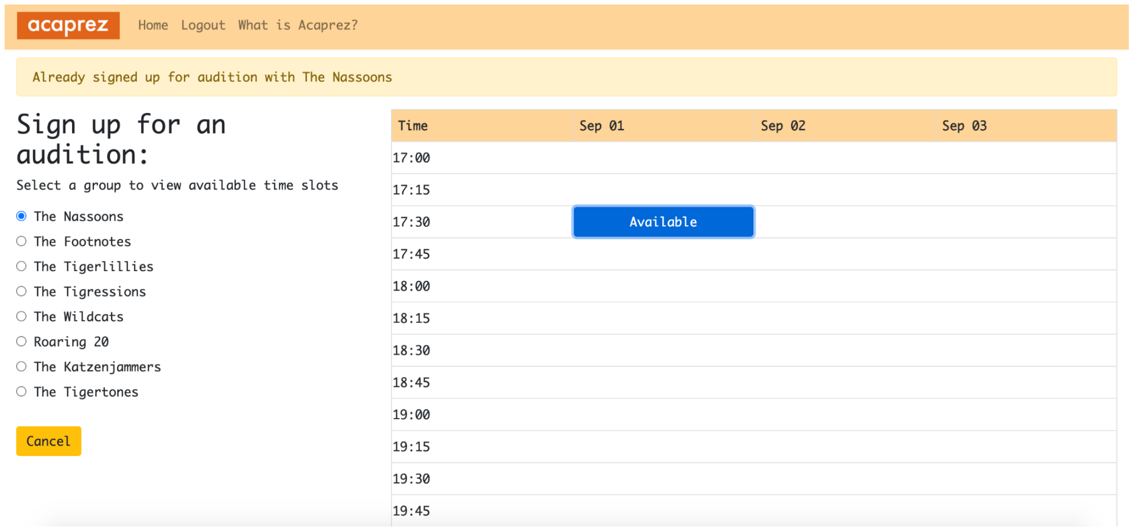
Sign up for an audition:

Select a group to view available time slots

- ● The Nassoons
- ○ The Footnotes
- ○ The Tigerlillies
- ○ The Tigressions
- ○ The Wildcats
- ○ Roaring 20
- ○ The Katzenjammers
- ○ The Tigertones

Time	Sep 01	Sep 02	Sep 03
17:00			
17:15			
17:30	Available		
17:45			
18:00			
18:15			
18:30			
18:45			
19:00			
19:15			
19:30			
19:45			

- Error banner when trying to sign up for a second audition



Already signed up for audition with The Nassoons

Sign up for an audition:

Select a group to view available time slots

- [✓ The Nassoons]
- The Footnotes
- The Tigerlillies
- The Tigressions
- The Wildcats
- Roaring 20
- The Katzenjammers
- The Tigertones

Time	Sep 01	Sep 02	Sep 03
17:00			
17:15			
17:30	Available		
17:45			
18:00			
18:15			
18:30			
18:45			
19:00			
19:15			
19:30			
19:45			

2. Match between system and the real world

The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

The language used within the site is consistent with that of an acapella scheduling platform, and we follow the conventions of the Acaprez audition cycle.

First, our site is tailored to the Acaprez association, so we've only included acapella groups at Princeton University that fall under their regulation: The Nassoons, The Footnotes, The Tigerlilies, The Tigressions, The Wildcats, Roaring 20, The Katzenjammers, and The Tigertones. Also following Acaprez conventions, each auditionee can sign up for an audition slot for any number of groups (but only one audition per group, with no time conflicts across groups), but can only accept a maximum of two callbacks. We used the term “callback” rather than a more vague but accessible term like “second round of audition” because this term is more specific in the context of acapella, and indicates a different style of audition. We also only let auditionees indicate their callback availability once, and not edit it, in line with Acaprez guidelines.

Another convention we used is when an auditionee is prompted to create a profile, we include the field for “voice part,” which is the melody carried by a particular part or voice when performing music. However, we made this field optional in case auditionees were unsure of their voicepart or did not fully understand what a voice part was.

Our site also supports the logistical order of the acapella audition cycle. Auditionees first sign up for audition slots as offered by the groups, then groups make their decisions about callbacks, and only then do auditionees have the option to add callback availability and accept callbacks. Auditionees cannot indicate callback availability until they have received at least one callback.

Another interactive example of a match between real world auditions and our site is that only auditionees can cancel an audition they've signed up for, while a group can only cancel an audition that no one has signed up for (i.e. a group can't cancel an audition slot that already has an auditionee associated with it).

3. User control and freedom

Users often choose system functions by mistake and near a clear "emergency exit." Support undo and redo.

One way we ensure user control is by giving the option to cancel/undo a number of actions. On the auditionee side, auditionees are able to cancel auditions that they no longer want to participate in, with a clear red cancel button next to each slot in their Current Auditions table. Acaprez does not allow auditionees to undo/redo certain actions, like edit callback availability or undo/redo callback acceptances, and so our site abided by this convention. Once callback availability is given and once callbacks are accepted, that information is binding. However, we use popups and warning messages to clearly communicate this to auditionees.

On the leader side, leaders are able to cancel audition slots that no auditionees have signed up for yet. As on the auditionee side, by Acaprez convention leaders cannot undo/redo offering callbacks out of fairness to auditionees. However, we use popups to clearly communicate this to leaders.

We also maintain a navigation bar, with options to return to the leader/auditionee home page as well as to log out. These options serve as a clear and consistent emergency exit for users. Additionally, when users are taken away from their leader/auditionee landing page, we include clear buttons that allow the user to return to their profile even without using the navigation bar.

4. Consistency and standards

Users shouldn't have to wonder whether different words/situations/actions mean the same thing

We maintained consistent terminology across our website. For example, we maintained the conventions of "auditions" versus "callbacks" for auditionees, leaders, and the admin page. We also matched action items with the tables they corresponded to. On the auditionee side, editing the "Current *Auditions*" table is done by clicking the "New *Audition*" button and editing the "*Callback* Offers" button is done by clicking "Add *Callback* Availability." On the leader side, the button names use similar conventions, with "Add *Audition* Times" changing the "All *Audition* Slots" table. Similarly, we use different tables to visually separate different information. It would be illogical to display callback and audition times in the same table, so we created separate tables to hold information about auditions and callbacks on both the leader and auditionee side.

More generally, we ensured that buttons with similar titles across pages, like "Return to Profile" would return either the auditionee or leader back to their landing page, and "Cancel" is placed directly next to or in line with the object it removes.

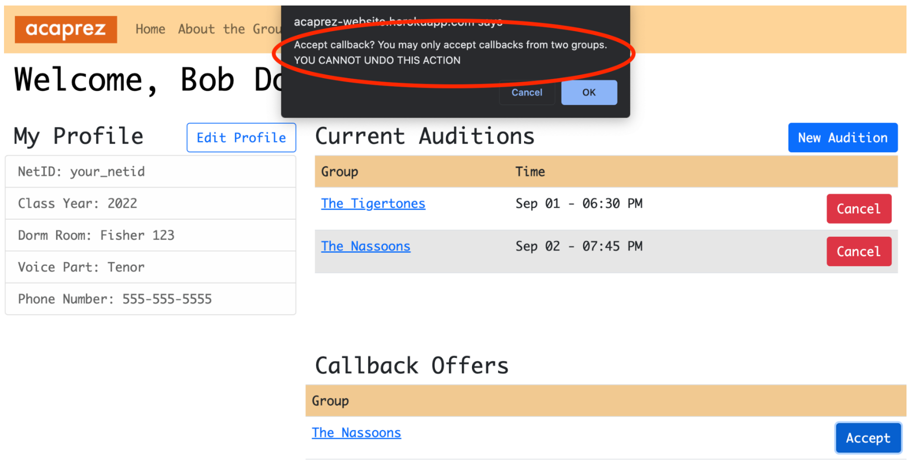
We also followed site design conventions like maintaining hyperlink text decoration, and employing colors like blue for buttons with positive actions (*add* audition, *add* audition times, *submit*) or red for buttons with negative actions (cancel). Green messages indicated successful actions, like successfully scheduling an audition, while a yellow message indicated an unsuccessful action, like attempting to schedule a second audition with one group.

5. Error prevention

Careful design that prevents problems from occurring in the first place

Our site uses a number of error prevention tools, including popups, form field validation, and confirmation pages.

- Popups to confirm intention of an action
  - We used pop ups so users would have to confirm actions to ensure they did not accidentally commit an action. For example, we used a popup to confirm auditionee audition sign ups, callback offering from leaders, callback accepting from auditionees, and, most importantly, before site resetting by the admin. The popups are especially important for actions that cannot be undone, like resetting the database and callback offering and accepting.



acaprez-website.herokuapp.com says

Accept callback? You may only accept callbacks from two groups. YOU CANNOT UNDO THIS ACTION

Welcome, Bob Do

My Profile

NetID: your_netid
Class Year: 2022
Dorm Room: Fisher 123
Voice Part: Tenor
Phone Number: 555-555-5555

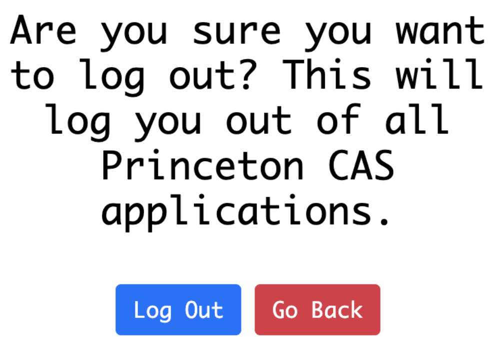
Current Auditions

Group	Time	
The Tigertones	Sep 01 - 06:30 PM	Cancel
The Nassoons	Sep 02 - 07:45 PM	Cancel

Callback Offers

Group	
The Nassoons	Accept

- User profile form validation
  - As discussed in our testing section, we used form validation to ensure that auditionee profiles fit a certain format. For example, our database only supports each field to hold 50 characters, so we applied a character maximum for all fields. We also applied validation for phone numbers to fit a “1234567890” or “123-456-7890” format.
- Logout confirmation page
  - We added in a logout confirmation page that confirmed a user intended to logout, which is important considering the logout button is within the navigation bar (and could likely be clicked by accident). The confirmation page also explains that logging out will log you out of CAS for a given session. It gives a user an emergency exit, returning to the leader or auditionee landing page upon clicking the “Go back” button.


Are you sure you want to log out? This will log you out of all Princeton CAS applications.

6. Recognition rather than recall

Making objects, actions, and options visible. Instructions should be visible and retrievable.

Our application uses information displayed in tables and a responsive user interface to make objections, actions, and options more visible across pages.

On the auditionee side, we use three separate tables to display current auditions (with groups and time slots), callback offers, and accepted callbacks to visibly track the auditionee's auditions and callbacks rather than putting that responsibility on them. Furthermore, auditions with conflicting times are removed from the "Sign Up for an Audition" table, so auditionees do not have to remember other auditions they signed up for. The auditionee profile information can also be found on the landing page for ease of editing.

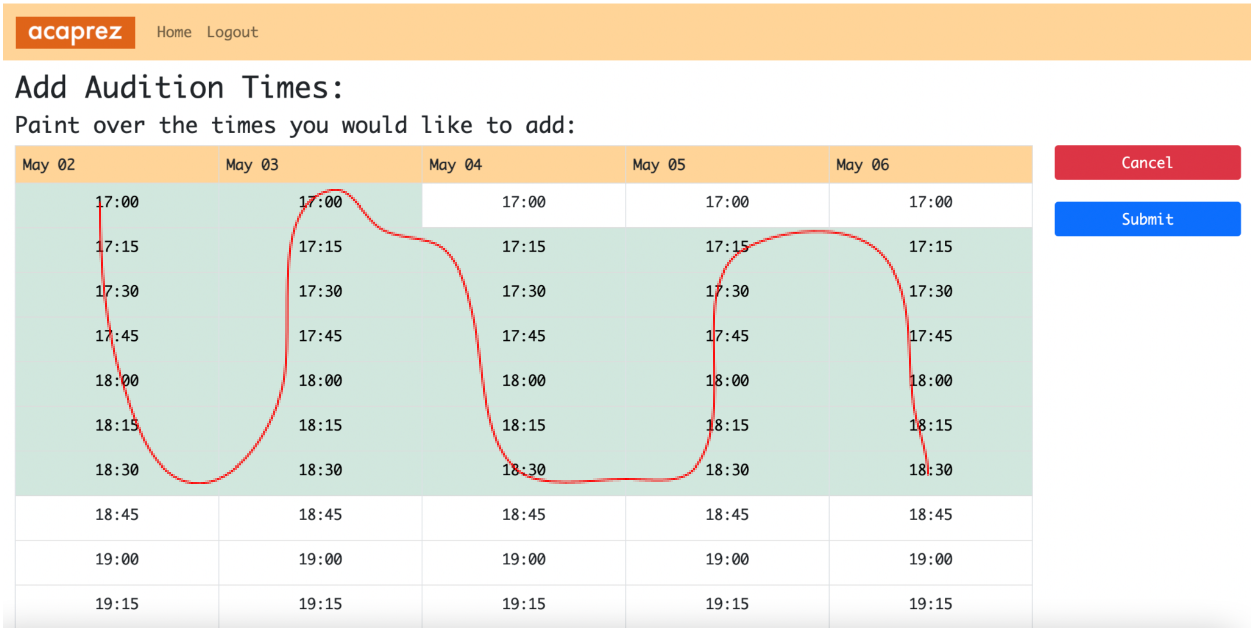
On the leader side, when adding audition times, we visually block out audition times with a gray "Already Scheduled" label. We also maintain three separate tables on the landing page that display all audition slots, pending auditions, and offered callbacks. This helps groups remember how many callbacks they've offered, and allows them to see which auditionees they have versus have not offered callbacks to.

7. Flexibility and efficiency of use

Allow users to tailor frequent actions

The part of our project that needed the most repeated actions is creating audition times from the leader's side. Initially we had individual checkboxes for each time, which would have to be clicked one-by-one, however during our own internal testing we found this to be tedious, particularly when selecting a lot of times. We then adapted our UI to be similar to that of *whenisgood*, a popular scheduling website. This enables the user to click and drag over multiple time slots, thus selecting as many time slots as necessary in one action.

Below is a diagram showing the motion of the cursor (with the mouse button held down), and how it selects 32 times in a couple of seconds.



Add Audition Times:

Paint over the times you would like to add:

May 02	May 03	May 04	May 05	May 06
17:00	17:00	17:00	17:00	17:00
17:15	17:15	17:15	17:15	17:15
17:30	17:30	17:30	17:30	17:30
17:45	17:45	17:45	17:45	17:45
18:00	18:00	18:00	18:00	18:00
18:15	18:15	18:15	18:15	18:15
18:30	18:30	18:30	18:30	18:30
18:45	18:45	18:45	18:45	18:45
19:00	19:00	19:00	19:00	19:00
19:15	19:15	19:15	19:15	19:15

With our login system, we also ensure that auditionees only have to create their profile once, when they first access the system. After that, they still have the ability to edit their profile, but it does not need to be completed with each login.

8. Aesthetic and minimalist design

Dialogues should not contain information which is irrelevant

Our application balances the amount of necessary information displayed to the user at one time. For the entire site, we maintained a minimal color scheme, using Bootstrap's automatic button colors depending on button intention (i.e. blue for primary buttons, red for warning buttons). Outside of these colors, we used black, white, gray, and a light orange to align with the Acaprez's orange logo. Beyond the choice of orange for navigation bar and table decoration, the other colors employed on the site are purposeful. As previously discussed, colors like green signified successful actions, yellow was used to highlight important profile and group information, and button colors aligned with button action. We also used striped tables to distinguish adjacent objects in the same table. Our choices for user input were also intentional. When signing up for an audition, it would be overwhelming to be able to see multiple groups' audition times at once, so we used radio buttons to ensure only one group could be selected at a time.

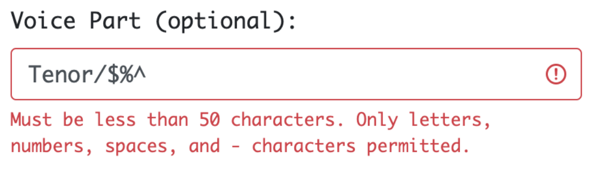
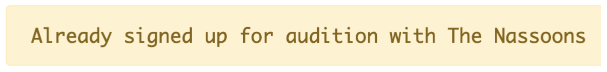
On the auditionee side, we used separate tables for information organization, but also ensured that information could move between tables. For example, once an auditionee accepts a callback, it moves from the "Callback Offers" table to the "Accepted Callbacks" table. Not only does this improve clarity and organization, but it also ensures that this callback only appears once on the auditionee page, rather than in both tables. When signing up for an audition, the large blue "Available" buttons are clear and easy to select to sign up for an audition.

On the leader side, we used an interactive "painting" design to facilitate leaders selecting audition slots. Initially, we had a checkbox for each audition slot, but it was very tedious for leaders to click every single slot. With the painting interaction, they can drag the cursor across many slots quickly, or individual clicking can be used. The slots also change color to visually cue that they are selected versus not selected. Like on the auditionee side, the leader side also has three separate tables that allow information to move between them. We ensured these tables would vertically scroll to constrain them to a certain height to limit overall page height for readability and organization. The “Pending Auditions” table holds all auditionees that signed up for an audition, but they are moved to the “Offered Callbacks” table if offered a callback. This visually separates the auditionees with and without callbacks.

9. Help users recognize, diagnose, and recover from error

Error messages should be expressed in plain language

We wanted to ensure that our error messages were both understandable and descriptive.

Error Message	Trigger
 Voice Part (optional): Tenor/$%^ Must be less than 50 characters. Only letters, numbers, spaces, and - characters permitted.	This error message was triggered by our form validation. We ensured the error message, which has to apply to any incorrect input, clearly stated our parameters for an acceptable form input.
 Already signed up for audition with The Nassoons	This error message was triggered when an auditionee attempted to sign up for multiple auditions with the same group. It clearly indicates with which group it occurred, and why the second audition sign up was not confirmed.

There are other similar error messages throughout our site, dealing with audition sign ups and form validation. All of them clarify why the error occurred, hopefully providing the user with enough context so as to not repeat the error. In the future, we hope to make error handling more robust as we work with acapella groups over the summer to prepare the site for fall auditions.

10. Help and documentation

Provide help and documentation. Information should be listed in concrete steps.

We hope that our users will not require help and documentation in addition to the descriptions throughout our site. However, if they would like to contact us, our names are listed at the bottom of the splash page, and they can reach us through our Princeton emails if they have any major issues or feedback. We didn’t think that our site required a tutorial since the majority of Princeton acapella students are familiar with the process of signing up for auditions, but in the future we would consider adding this section if user feedback prompted it.

Appendix A: Task List

Introductory Questions

1. What class year are you in?
2. Are you in an acapella group? If so, are they part of Acaprez?
3. Have you ever auditioned for an acapella group?
4. Are you interested in auditioning for an acapella group in the future?

Main Tasks

Note: our site was preloaded with example audition times from select groups and example auditionees to facilitate the testing process. We would initially only read the task description and required information (like group netid), rather than all details in the bullets.

Task 1: Logging in as a group leader

1. Navigate to https://acaprez-website.herokuapp.com/.
2. Login with the developer login, and enter “nassoons” in the input field.

Task 2: Adding audition times

1. Click the blue “Add Audition Times” button.
2. Paint over any audition times you want.
3. Click the blue “Submit” button once satisfied.

Task 3: Logging out as a leader

1. Click the “Logout” button in the navigation bar at the top of the leader landing page.
2. Click the blue “Logout" button on the confirmation page.

Task 4: Logging in as an auditionee

1. Navigate to https://acaprez-website.herokuapp.com/.
2. Login with CAS using their personal netid.
3. Set up a user profile, since this will be their first time using the site.

Task 5: Signing up for auditions as an auditionee

1. Click on the “What is Acaprez?” button in the navigation bar to learn more about the different groups. Return to profile when done.
2. Once back on the auditionee landing page, click the blue “New Audition” button
3. Sign up for at least 3 auditions, but make sure one of the groups is the Nassoons.
4. Click the “Cancel” button to return back to the auditionee landing page.

Task 6: Logging out as an auditionee

1. Click the “Logout” button in the navigation bar at the top of the auditionee landing page.
2. Click the blue “Logout" button on the confirmation page.

Task 7: Offering callbacks as a leader

1. Login as the leader of The Nassoons as previously specified.
2. Click on the links on the auditionees that have Pending Auditions to reach their auditionee profile.
3. Once you’ve learned about the auditionees, offer callbacks to at least yourself by clicking the blue “Offer callback” button next to your (and any other auditionees’) name(s)..

Task 8: Scheduling and accepting callbacks as an auditionee

1. Login as an auditionee as previously specified
2. Click the “Add Callback Availability” and enter times you would be available. You can only enter this information once.
3. If you would like to accept a callback from a group, click the “Accept” button to the right of the offered callback. You can only do this for two groups, and this action cannot be undone.

Task 9: Logging in as an admin

1. Navigate to https://acaprez-website.herokuapp.com/.
2. Login with the developer login, and enter “admin” in the input field.

Task 10: Admin sets new audition/callback dates + resets the site for a new cycle

1. Click the gray input box under “Select audition dates”, and with the calendar input form, pick any 3 audition dates.
2. Next, click the gray input box under “Select callback date(s)”, and with the calendar input form, pick a callback datetime.
3. Using the “Add Row” button, add 3 more callback datetimes.
4. Reset the site with your inputted dates.

Key Takeaway Questions

1. What was the most difficult task you had to complete?
2. What would you change about the user interface?
3. What felt intuitive? What felt confusing?

Appendix B: User Interview Notes

Structure

1. Ask intro questions
2. Read only the task titles, not the bullets! Bullets are too descriptive
3. Ask takeaway questions/get descriptive feedback on what to improve

Interviews

Interview A

- Introduction question answers:
    - 2022
    - No
    - No
    - No
    - Device: Mac
- Task notes:
    - Had trouble with the phone number
    - Otherwise, didn't have any trouble with any of the tasks
- Takeaways:
    - The hardest part was visualizing the callback and audition dates without being able to look at the same calendar.
    - Could've picked a better color for the UI, feels cliche
    - Make it easier to directly decline a callback, rather than ghosting them
    - The login, the audition sign up, and the group selection, and the profile creation all felt intuitive

Interview B

- Introduction question answer:
    - 2025
    - No
    - No
    - No

      Device: PC
- Task notes:
    - Painting over doesn't seem intuitive, people just click
    - Hyphens in phone number are troublesome
    - "Cancel" instead of return home is confusing

    - Callback dates needing to add rows while audition dates don’t is confusing → makes it hard to add multiple callback datetimes
- Takeaways:
    - Selecting callback dates was the most difficult because it was different from audition dates
    - Changing navbar is confusing (e.g. logout button is only there sometimes)
    - Looking at the buttons, could definitely figure everything out

Interview C

- Introduction questions:
    - 2024
    - No
    - No
    - Yes
    - Device: iPhone
- Task notes
    - Zooming in while tapping audition times on the phone
    - People were confused what voice part meant
    - Phone number hyphens were difficult
    - Add row was again confusing on the phone because the calendar was different from the earlier one.
- Takeaways
    - The view on the phone was difficult because everything was very small, especially because it was her first time using the site
    - Was intuitive with more direction; wasn’t hard to find anything
    - Would change the Add Row button
    - The login section was intuitive

Interview D

- Introduction questions:
    - 2024
    - No
    - No
    - No
    - Device: Mac
- Task notes
    - Why the dorm room?
        - Also just building, or actual room number
    - What is Acaprez? link isn’t clear what it actually means
        - Say something about group info
        - Actual info on it is good

        - Confused by the fact that the selection for audition days included all the days in one calendar, but not for callback days
            - Didn't understand the add row until explained
- Takeaways
    - Most difficult task
        - Selection for the callback dates
            - If there's a lot of times, it is kind of tedious
        - Everything else was easy to understand
    - UI Changes
        - Clear up the What is Acaprez? Confusion
        - Change add row button to blue or other more attention drawing color
    - Intuitive
        - Setting audition times
        - Choosing audition times
    - Confusing
        - Adding callback days
        - Where to find information on groups

Interview E

- Introduction question answers:
    - 2024
    - Yes
    - Yes
    - No
    - Device: Mac
- Task notes:
    - Just clicked timeslots 1-by-1 rather than painting over them
    - Didn't put dashes in phone number
    - Didn't realize to press cancel after signing up for auditions
    - Didn't change time for callback sessions
- Takeaways:
    - Phone number was most difficult, should auto insert the dashes
    - Didn't see that there was a time option for the datetime picker
    - Nothing felt confusing, overall pretty intuitive
    - Preferred and would rather use this system than the previous way of signing up for auditions

Interview F

- Introduction question answers:
    - 2024
    - No
    - No
    - No

    - Device: Mac
- Task notes:
    - Successfully painted over range of times
    - Accidentally didn’t enter year correctly, fixed it after reading error message
    - Went back to fix an error on confirm profile screen
    - Successfully added multiple callback times without guidance
- Takeaways:
    - Selecting audition dates/timeslots on the admin-side was the least clear, but still navigated successfully without guidance
    - Login, setting audition slots, signing up for an audition, offering/accepting callback all felt intuitive

Interview G

- Introduction questions
    - 2024
    - Yes
    - Yes
    - Yes
- Task notes:
    - Tasks felt intuitive
    - Site was easy to understand without task descriptions
    - Was a lot better than the old way
    - Appreciated ability to sign up for different groups in the same place
- Takeaways
    - Add locations for auditions, make groups able to edit their profile cards
    - Nice for callback control, since it’s impossible to accept more than two callbacks

Interview H

- Introduction question answers:
    - 2022
    - Yes
    - Yes
    - No
    - Device: Mac
    - Former president of a group, has been involved in scheduling auditions and callbacks
- Task notes:
    - Tried to input multiple callback session times using the same datetime picker, and took a second to understand how it works.
- Takeaways:
    - Scheduling callbacks was most difficult, suggested adding instructions or a confirm button to the datepicker to make it more obvious.

- Didn’t love the browser confirmation windows, felt a little out of theme and would also be easier if a popup appears nearer the button you just clicked.
- Everything else was very intuitive.
- Prefers to the current system, and given Wase is expiring thinks it would be a good alternative.
- Doesn’t have faith in Acaprez being organized and having the motivation to actually implement it.
- Stressed the importance of the groups coming together in person to schedule the auditions, so might be useful to ensure multiple people have access to the availability table in the admin view.