Automated Lab Room Allocation System (ALRAS) Application Documentation

Prepared by: Sourav Purification
Application type: Web Application
Framework: Django MVT Framework
Project Source-Code Repository: https://github.com/soupurtion/alras
Project Tracker: https://github.com/users/soupurtion/projects/3
Documentation: https://github.com/soupurtion/alras/tree/main/Documentation
Version History:

Version	Date Created	Last Modified
v1.0	11/01/2023	11/04/2023

Table of Contents

Part 1: Project Introduction

1.1 About the ALRAS application

Problem:

The Cyber Security Innovation Center (CSIC) faces challenges in efficient allocation of lab rooms to its students. The current manual reservation process is cumbersome and time-consuming for both the lab coordinator and the cyber security students. This inefficiency results in delays, conflicts, and a lack of real-time room availability information. How can I design and implement an automated lab room allocation system that enables students to check real-time room availability, request slots without coordinator approval where applicable, and efficiently manage lab room reservations?

Solution:

I design and build a web application to solve the allocation of lab rooms to the students by replacing manual email based reservation to web application based self allocation system. In the application the students will login with their access credentials. After logging in they can reserve, cancel or update the labroom. The application also has a feature to display allocated slots, available slots without student login. Some labrooms require additional approval from the lab coordinator which is also a feature of the application.

1.2 User Stories

The application requirements and customer stories are designed based on customer interviews. User stories are created based on the interviews. Each user story is broken down into tasks. Below are the user stories:

1. Display available slot: As a student I want an automated lab room reservation webportal so that I can see what rooms/slots are available within a period of time (a week) without logging in.
2. Current reservation status: As a student I want to see the current reservation status of a labroom so that I can find a suitable slot for me without logging in.
3. Add reservation: As a student I want to securely login using my username and password so that I can reserve a labroom/slot.
4. Edit reservation: As a student I want to edit my reservation after securely login using my username and password so that I can change my reservation status.
5. Delete reservation: As a student I want to delete my reservation after securely login using my username and password so that other students can use it.

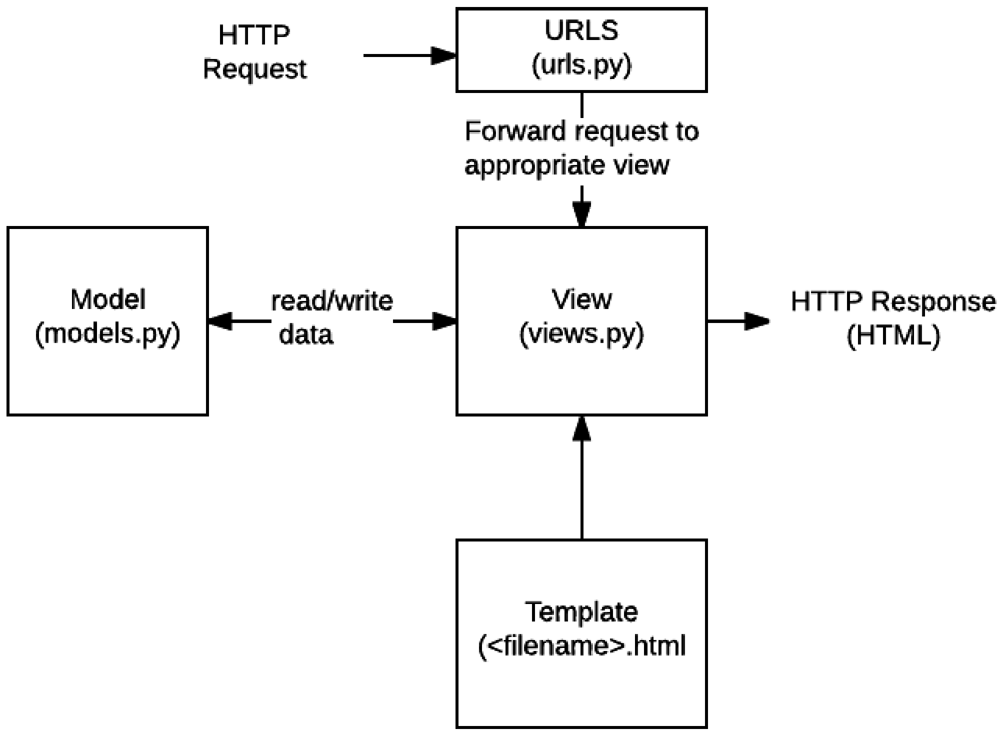
6. Approve reservation: As a lab coordinator I want to see any reservation that needs my approval so that I can take quick action.

1.3 Django MVT Framework

I use the Django MVT (Model, View, Template) framework to build the application. Before going deep into the application it is better to have a brief overview of the Django MVT framework.

The MVT architecture has three parts as the name suggests: Model, View, Template
Model: The model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database
View: The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files
Template: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted

The figure 1 shows different components of the django MVT framework. When a user requests any webpage using http request it directs to URLS (urls.py) of MVT and forwards the request to VIEWS (views.py) for the appropriate view. In the views.py classes needed to be defined for each https request to specific URLs defined in urls.py. It is the VIEW that coordinates among models (for data), templates (for html files that to be displayed in user webpage) and sends http response.



Sample Code Snippets:

urls.py
```
path('', views.index, name='index'),
```

views.py
```
def index(request):
# Render the HTML template index.html with the data in the context variable.
    student_active_portfolios =
Student.objects.select_related('portfolio').all().filter(portfolio__is_active=True)
    print("active portfolio query set", student_active_portfolios)
    return render( request, 'portfolio_app/index.html',
{'student_active_portfolios':student_active_portfolios})
```

index.html
```
<h1>Computer Science Portfolios</h1>
<h4>List of Students with Active Portfolios</h4>
```

```
{% for i in student_active_portfolios %}
<div class="col-sm-4 p-1 border rounded-top">
   <div class="container-fluid-sm">
   Name: {{i.name}}
   <p><strong>Portfolio: </strong>{{ i.portfolio.title }}
   <a class="btn btn-primary" href="{{ i.portfolio.get_absolute_url }}"
role="button">View</a>
   </div>
   </div>
{% endfor %}
```

Explanation:
When a user requests for the index.html (the home page or default webpage that the server returns) then the urls.py file matches the url requests by the user. If the url requested by the user is specified in urls.py then it forwards the request toward the view function views.index. In views.py the *index* function is defined. Here, the function fetches data from the student model and renders the web page 'portfolio_app/index.html' with the argument student_active_portfolios as a dictionary. The index.html now has all the required data and format to display contents to the user.

Part 2: Setting up the application

2.1 Application environment setup

Creating a virtual environment: In order to work with the django framework I need to set up the environment. I create and activate a virtual environment for python with the following commands. The purpose of the virtual environment is to make a separation between the base python installation in the system and the python used in the application.

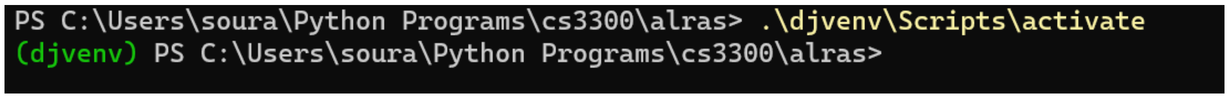
```
# go to the project directory
cd alras

# Run the command to create a virtual environment
python3 -m venv djvenv

#activate the python virtual environment
```

```
source djvenv/bin/activate #for linux machines
./djvenv/Scripts/actvate   # for windows machines

#The final snippets of virtual environment
```



```
PS C:\Users\soura\Python Programs\cs3300\alras> .\djvenv\Scripts\activate
(djvenv) PS C:\Users\soura\Python Programs\cs3300\alras>
```

```
# Install Django
pip install django

# upgrade the python pip package manager
python3 -m pip install --upgrade pip

# Install bootstrap
pip install django-bootstrap-v5
```

With this the environment setup is completed and our django application is ready to install which is described in the section 2.2.

2.2 Installing the basic application

Installing the alras application:
With the python virtual environment activated and django installed, the basic alras application is installed with the following commands:

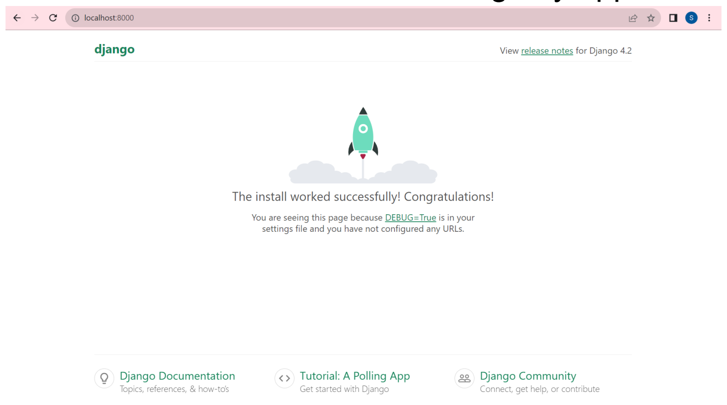
```
#install the alras django project
django-admin startproject alras

# Reorder the directory for the ease of application use
mv alras/manage.py
./ mv alrast/alras/* alras
rm -r alras/alras/


# Create requirements file to save what is installed to run the application
pip freeze > requirements.txt

# run the server for testing
python manage.py runserver
```

```
# The basic server without installing any application
```



The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Creating the alras_application app
Once the django project is installed then the alras_application is created on which the web application resides.

Step 1:

```
#Create Application
django-admin startapp alras_application
```

Step 2:
Open the alras/settings.py file and add the alras_application to the installed apps and add support for authenticating users:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bootstrap5',
    'alras_application',
]
AUTHENTICATION_BACKENDS = [
    'django.contrib.auth.backends.ModelBackend',
]
```

2.3 Github version control

To manage the versions of the project, the following github repository is used for this project. For each sprint a new branch is created and once the branch is ready then it is merged with the main.
Github repository: https://github.com/soupurtion/alras
To work with the github following are some useful commands that I need more frequently:

commit: This command is used to confirm the changes in the local repository. An argument “-m” is used to add a description of the commit.
Example: git commit -m "Added new feature"

pull: This command fetch changes from the remote repository (e.g. stored in GitHub) and merges them into the local repository. This command merges two commands: git fetch and git merge. It takes the branch name as an argument.
Example: git pull origin main

push: This command does the opposite of the pull command. It sends the locally committed code files into the remote repository if the local repository is linked to a remote repository. It takes the branch name as an argument.
Example: git push origin main

Add: This command adds a file or a folder into the local repository. Before committing changes this command needs to run.
Example: git add test_file.py

clone: This command is used to create a copy of a remote repository into a local machine.
Example: git clone https://github.com/Roodcube/cs3300DjangoFandango

status: It shows the current status of the working directory and stages the changes
Example: git status

log: This command shows a log of commits in the repository
checkout: This command use to switch to a different branch or commit.

Sample commands:

```
# Cloning the github repository
git clone https://github.com/soupurtion/alras

# Adding all files into the github project
git add .

# Committing changes
git commit -m “First commit”

# Upload/push committed changes to the remote repository i.e. github
git push

# Adding any local project to the remote repository
git remote add origin https://github.com/soupurtion/alras

#checking git status
git status

# Creating a new branch
git branch Sprint-01
git checkout Sprint-01

#merging the branch Sprint-01 to main
git checkout main
git merge Sprint-01
```

Part 3: Understanding the skeleton

In this part I describe the basic building blocks: models, views and templates used to build the sprint 01 of the project. The summary relationship of urls, models, views, and templates are given at first and then detail of each component is described.

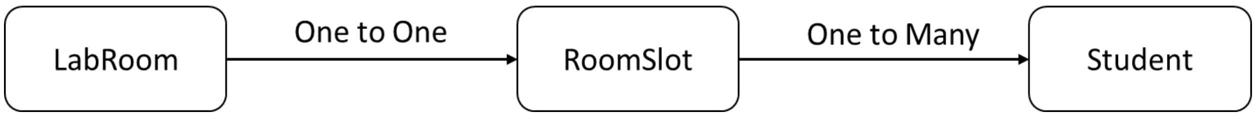
User requirement	Url path in url.py	Function in views.py	Model in model.py	Html in template
`Homepage with reserved slots`	`path('', views.index, name='index')`	`index(request)`	Student	index.html

List view of labrooms	path('labroom/', views.LabRoomListView.as_view(), name='labroom')	LabRoomListView(generic.ListView)	LabRoom	labroom_list
Detail view of each labroom/slot	path('labroom/<int:pk>', views.LabRoomDetailView.as_view(), name='labroom-detail')	LabRoomDetailView(generic.DetailView)	LabRoom, Student, RoomSlot	labroom_detail
Reserve slot	path('labroom/<int:pk>/reserve_slot/', views.reserveSlot, name='reserve_slot')	reserveSlot(request, pk)	LabRoom, Student, RoomSlot	reserve_slot.html
Cancel Slot	path('labroom/<int:pk>/cancel_slot/', views.cancelSlot, name='cancel_slot')	cancelSlot(request,pk)	Student, RoomSlot	cancel_slot.html

`Update Slot`	`path('labroom/<int:pk>/update_slot/', views.updateSlot, name='update_slot')`	`updateSlot(request, pk)`	LabRoom, Student, RoomSlot	update_slot.html

3.1 Models

In this project I create three models: Student, LabRoom, RoomSlot. These models have relationships among them as below:



The Code Snippets with explanation in comments for all of the three models are given below:

```
# Each Room has three slots for each day. In RoomSlot model
three slots # are defined and each of the slots will be
assigned to a room. Hence
# any room can have these three slots. Each room can have only
one slot # one time.

class RoomSlot(models.Model):
    SLOTS = (
        ('slot-1', '09:00 - 12:00'),
        ('slot-2', '12:00 - 15:00'),
        ('slot-3', '15:00 - 18:00')
    )
    slot = models.CharField(max_length=200, choices= SLOTS)
    title = models.CharField(max_length=200)
    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('slot-detail', args=[str(self.id)])

# Each Student model has student information. A single student 
can
# reserve only one slot. Hence a slot in the student model has 
one to   # many relationships. All of the fields in the student 
model are        # mandatory.

class Student(models.Model):
    MAJOR = (
        ('PhD-CS', 'PhD in Computer Science'),
        ('PhD-Sec', 'PhD in Security'),
        ('MS-CS', 'MS in Computer Science'),
        ('MS-Sec', 'MS in Security'),
    )
    name = models.CharField(max_length=200)
    email = models.CharField("UCCS Email", max_length=200)
    major = models.CharField(max_length=200, choices= MAJOR)
    slot = models.ForeignKey(RoomSlot, 
on_delete=models.CASCADE, unique=True, default = None)
    purpose = models.CharField(max_length=200)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('student-detail', args=[str(self.id)])

# In the LabRoom model each room has a one to one relationship
```

```
with the RoomSlot model using the field: slot.

class LabRoom(models.Model):
    ROOMS = (
        ('101-A', 'Room 101-A'),
        ('102-A', 'Room 102-A'),
        ('103-B', 'Room 103-B'),
        ('104-B', 'Room 104-B'),
        ('105-C', 'Room 105-C'),
    )
    title = models.CharField(max_length=10,choices=ROOMS)
    slots = models.OneToOneField(RoomSlot,
on_delete=models.CASCADE, unique=True, default=None)

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('labroom-detail', args=[str(self.id)])
```

Explanation:
RoomSlot model: Each Room has three slots for each day. In the RoomSlot model three slots are defined and each of the slots will be assigned to a room. Hence any room can have these three slots. Each room can have only one slot one time.
Student model: Each Student model has student information. A single student can reserve only one slot. Hence a slot in the student model has one to many relationships. All of the fields in the student model are mandatory.
LabRoom model: In the LabRoom model each room has a one to one relationship with the RoomSlot model using the field: slot.

3.2 Views

In the sprint-01 view functions are defined for displaying the list of items, detailed view of each item, adding a new entry, deleting the entry and updating the entry.

Displaying the homepage (index.html): In the home page, a list of students who already booked a slot is displayed.

```
def index(request):
    student_active_today = Student.objects.all()
    return render( request,
'alras_application/index.html',{'student_active_today':student_
active_today})
```

The information about the list of students is retrieved using the model manager: `student_active_today = Student.objects.all()`. This model manager simply returns all the students who reserved a slot. The index.html file uses the list of students to display the data.

Display list of items: I used a generic ListView function to display list of labroom slots as below:

```
class LabRoomListView(generic.ListView):
    model = LabRoom
```

The listView function gives a list of labroom slots inherently with a labroom_list variable that is passed to the template.

Display details of an item: To display the details of each item I use the DetailView function of LabRoom class. However, according to the user requirement the detail view contains the student information that reserved the slot. Hence, an additional get_context_data function is defined which returns the student and slot information.

```
class LabRoomDetailView(generic.DetailView):
    model = LabRoom
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        #print(self.objects.values().all())
        a =
LabRoom.objects.filter(id=self.kwargs['pk']).values().all()[0][
'slots_id']
```

```
        context["roomslot"] = RoomSlot.objects.filter(id=a)
        context["students"] = Student.objects.filter(slot_id=a)
        return context
```

Reserve a slot: To reserve a slot a form is created with a student model. A student can reserve a slot if the slot is available for reservation. When a student enters the detailed view of the room/slot then the reservation option will appear. The form for the reservation is given below:

forms.py

```
from django.forms import ModelForm
from .models import Student, LabRoom, RoomSlot
#create class for project form
class ReserveSlotForm(ModelForm):
    class Meta:
        model = Student
        fields =('name', 'email', 'major','purpose')
```

The view function for the reservation: The slot is already selected when the student selects the detail view of the slot.

```
def reserveSlot(request, pk):
    slot_id =
LabRoom.objects.filter(id=pk).values().all()[0]['slots_id']
    form = ReserveSlotForm()
    roomslot = RoomSlot.objects.get(id=slot_id)
    print(roomslot)

    if request.method == 'POST':
        # Create a new dictionary with form data and slot_id
        student_data = request.POST.copy()
        student_data['slot_id'] = slot_id
        form = ReserveSlotForm(student_data)

        if form.is_valid():
            # Save the form without committing to the database
            student = form.save(commit=False)
            # Set the slot relationship
            student.slot = roomslot
            student.save()

            # Redirect back to the portfolio detail page
            return redirect('labroom-detail', slot_id)

    context = {'form': form,'roomslot':roomslot}
    return render(request,
'alras_application/reserve_slot.html', context)
```

Cancel the reservation:
The view function to cancel the reservation is given below. When the student wants to cancel the reservation then a prompt will appear for confirmation. In addition to that, the cancelSlot view function redirects to the detail view of the slot.

```
def cancelSlot(request,pk):
    slot_id =
LabRoom.objects.filter(id=pk).values().all()[0]['slots_id']
    roomslot = RoomSlot.objects.get(id=slot_id)
    student = Student.objects.get(slot_id=slot_id)
    if request.method == 'POST':
        student.delete()
        return redirect('labroom-detail', slot_id)

    return
render(request,'alras_application/cancel_slot.html',{'pk':pk,'r
oomslot':roomslot})
```

Update the slot: Updating slot is similar to the reserve slot with an expectation that the form will load with existing data and the data will save in existing student id. The view function for the updating slot is below:

```
def updateSlot(request, pk):
    slot_id = 
LabRoom.objects.filter(id=pk).values().all()[0]['slots_id']
    student = 
Student.objects.filter(slot_id=slot_id).values()[0]
    s_id = student['id']
    print(s_id)
    roomslot = RoomSlot.objects.get(id=slot_id)
    form = ReserveSlotForm(initial=student)

    if request.method == 'POST':
        # Create a new dictionary with form data and slot_id
        student_data = request.POST.copy()
        student_data['slot_id'] = slot_id
        form = ReserveSlotForm(student_data)

        if form.is_valid():
            # Save the form without committing to the database
            student = form.save(commit=False)
            # Set the slot relationship
            student.slot = roomslot
            student.id = s_id
            student.save()

            # Redirect back to the portfolio detail page
            return redirect('labroom-detail', slot_id)

    context = {'form': form,'roomslot':roomslot,'pk':pk}
    return render(request, 
'alras_application/update_slot.html', context)
```

3.3 Templates

To display the contents of the web pages that the users requested are stored in templates directory. Following the templates of the application in Spring 01. To make the web pages responsive bootstrap is used and added in the base template.

Base_template.html: The base template designs the basic structure of the webpage which includes bootstrap definition, navigation menu bar, uccs logo.

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
<title>CSIC</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width,
initial-scale=1" />
{% load bootstrap5 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
</head>
<body>
<div class="container-fluid">
<!-- Navbar -->
<nav class="navbar navbar-expand-lg bg-body-tertiary">
<div class="container-fluid">
<img src="{% static 'images/uccs_logo.gif' %}">
<button class="navbar-toggler" type="button"
data-bs-toggle="collapse" data-bs-target="#navbarNav"
aria-controls="navbarNav" aria-expanded="false"
aria-label="Toggle navigation"><span
class="navbar-toggler-icon"></span></button>
<div class="collapse navbar-collapse" id="navbarNav">
<ul class="navbar-nav">
<li class="nav-item">
<a class="nav-link active" aria-current="page" href="{% url
'index' %}">Home</a>
```

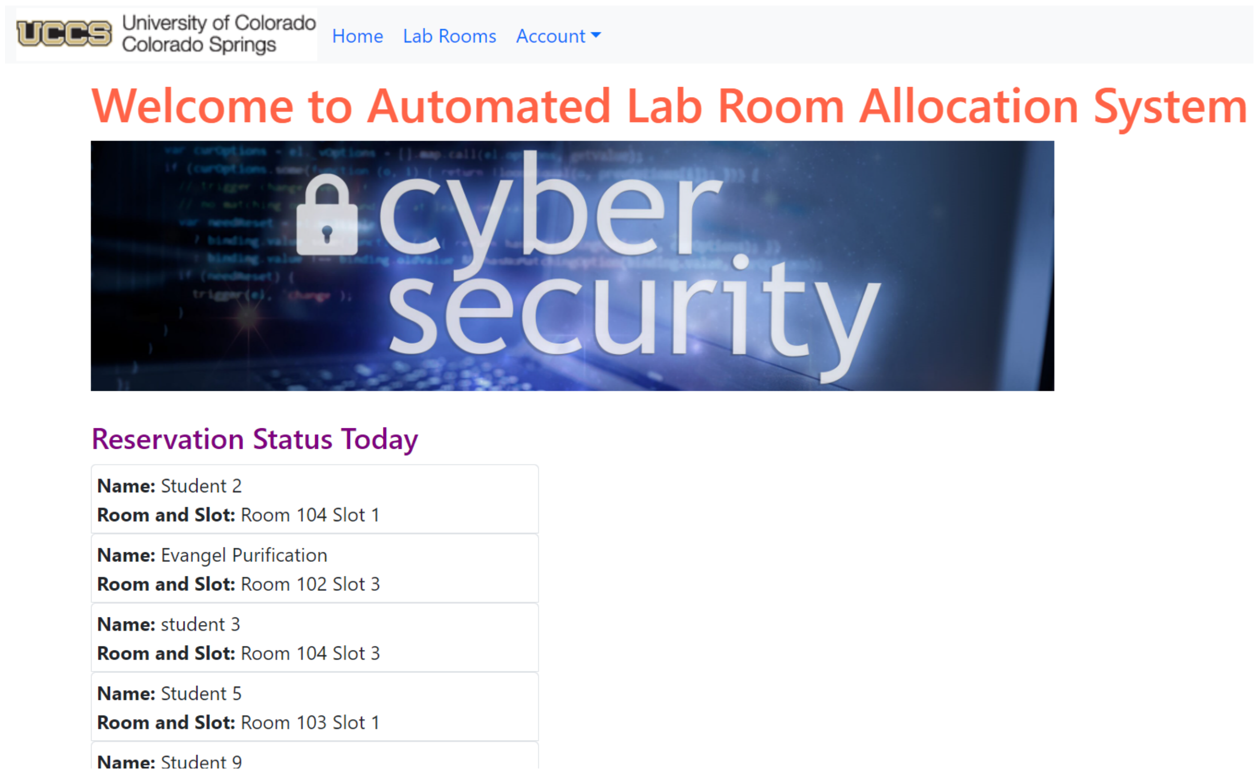
```
</li>
<li class="nav-item">
<a class="nav-link" href="{% url 'labroom' %}">Lab Rooms</a>
</li>
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" role="button"
data-bs-toggle="dropdown">Account</a>
  <ul class="dropdown-menu">
    <li><a class="dropdown-item" href="#">Sign In</a></li>
    <li><a class="dropdown-item" href="#">Sign Up</a></li>
  </ul>
</li>
</div>
</div>
</nav>
<!-- add block content from html template -->
{% block content %}
{% endblock %}
</div>
</body>
</html>
```

Index.html: The index html inherits the base_template and displays the student list who reserve slots. The index function defined in the view function provides necessary data for the index.html. It also includes a static image file.

```
{% block content %}
<div class="container">
  <div class="row justify-content-center">
<h1>Welcome to Automated Lab Room Allocation System</h1>
<img src="static/images/cyberLab_r.jpeg" class="img-fluid"
alt="Cinque Terre">
<h4>Reservation Status Today</h4>
</div>
</div>
```

```
{% for i in student_active_today %}
<div class="col-sm-3 p-1 border rounded-top">
    <div class="container-fluid-sm">
    <strong>Name:</strong> {{i.name}}
    </br>
    <strong>Room and Slot: </strong>{{ i.slot.title }}
    </div>
</div>
{% endfor %}
```

Index.html web page view



Welcome to Automated Lab Room Allocation System

Reservation Status Today

Name: Student 2
Room and Slot: Room 104 Slot 1

Name: Evangel Purification
Room and Slot: Room 102 Slot 3

Name: student 3
Room and Slot: Room 104 Slot 3

Name: Student 5
Room and Slot: Room 103 Slot 1

Name: Student 9

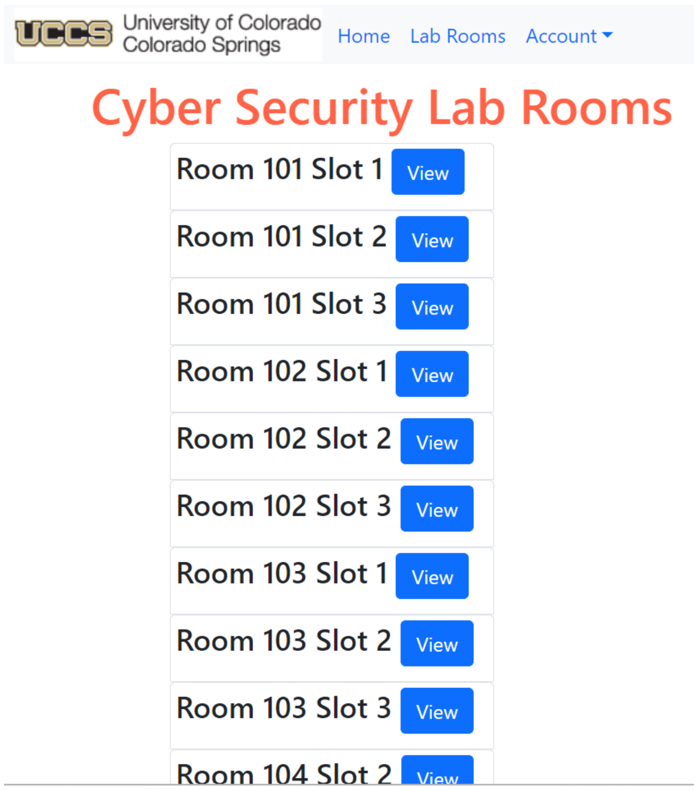
Labroom_list.html: This html template provides the list of labrooms. The ListView function in view.py provides the necessary data for building the contents.

```
<h1>Cyber Security Lab Rooms</h1>

{% if labroom_list %}
<ul>
```

```
{% for labroom in labroom_list %}
<ul>
<a href="{{ labroom.get_absolute_url
}}">{{labroom.slots.title}}</a>
</ul>
{% endfor %}
</ul>
{% else %}
<p>There are no lab rooms configured.</p>
{% endif %}
```

labroom_list .html webpage view



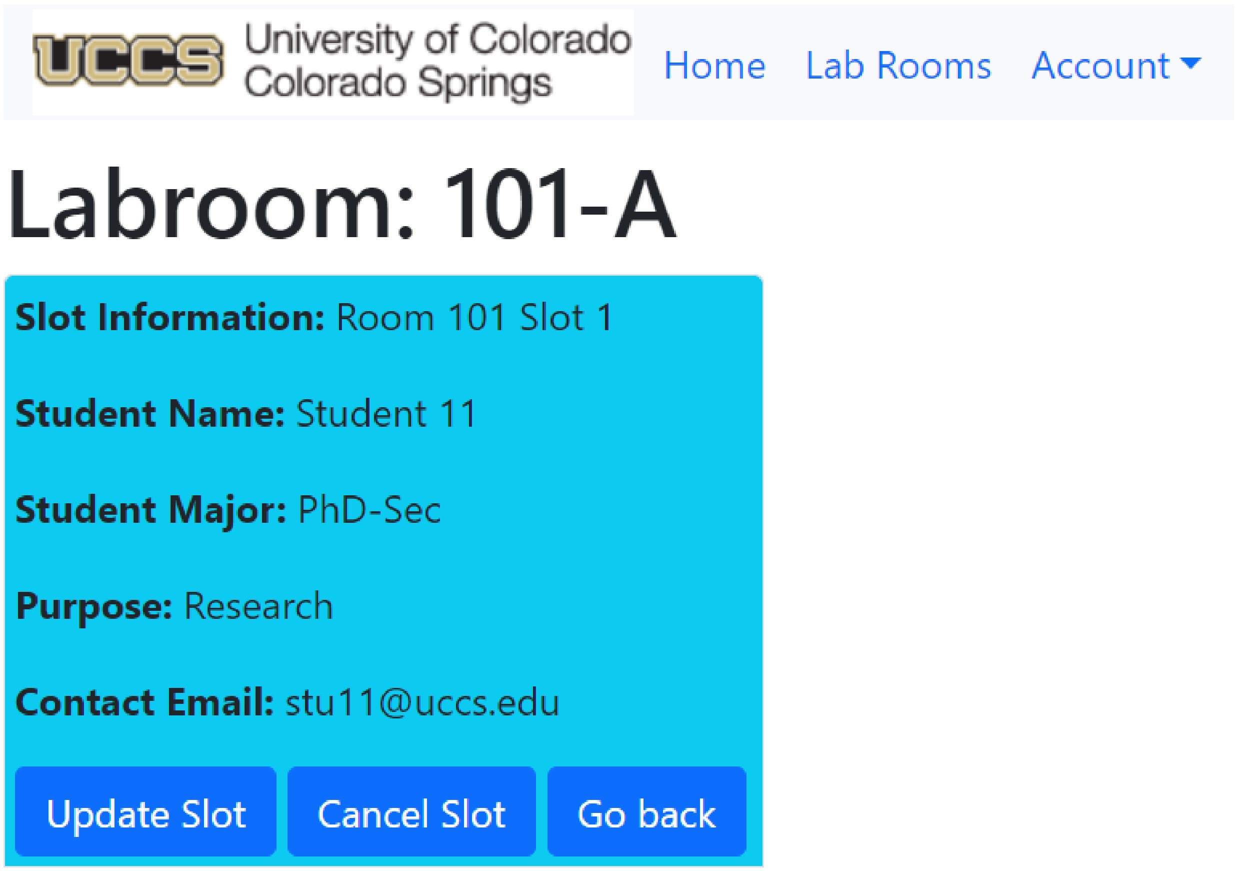
Cyber Security Lab Rooms

Room 101 Slot 1 View
Room 101 Slot 2 View
Room 101 Slot 3 View
Room 102 Slot 1 View
Room 102 Slot 2 View
Room 102 Slot 3 View
Room 103 Slot 1 View
Room 103 Slot 2 View
Room 103 Slot 3 View
Room 104 Slot 2 View

Labroom_detail.html: This html template provides the detail of labrooms. The DetailView function in view.py provides the necessary data for building the contents. If the slot is available for reservation then a Reserve Button is available, on the other hand if the slot is already booked then, an update and cancel button will be available. However, for each case the Go Back button is persistent.

```
<h1>Labroom: {{ labroom.title }}</h1>
{% for room in roomslot %}
<p><strong>Slot Information:</strong> {{ room.title }}</p>
{% endfor %}
{% if students %}
{% for student in students %}
<p><strong>Student Name:</strong> {{ student.name }}</p>
<p><strong>Student Major:</strong> {{ student.major }}</p>
<p><strong>Purpose:</strong> {{ student.purpose }}</p>
<p><strong>Contact Email:</strong> {{ student.email }}</p>
{% endfor %}
<a class="btn btn-primary" href="{%url 'update_slot'
labroom.id%}" role="button">Update Slot</a>
<a class="btn btn-primary" href="{%url 'cancel_slot'
labroom.id%}" role="button">Cancel Slot</a>
{% else %}
<p>This slot is available for reservation</p>
<a class="btn btn-primary" href="{% url 'reserve_slot'
labroom.id %}" role="button">Reserve</a>
{% endif %}
<a class="btn btn-primary" href="{%url 'labroom'%}"
role="button">Go back</a>
```

labroom_detail.html webpage view



Labroom: 101-A

Slot Information: Room 101 Slot 1

Student Name: Student 11

Student Major: PhD-Sec

Purpose: Research

Contact Email: stu11@uccs.edu

To reserve, update or cancel the selected slot templates are almost similar and the codes are self explanatory. The codes are given below:

reserve_slot.html

```
{% block content %}
<h4> Reserve Slot: {{ roomslot }} </h4>
<form action="" method="POST">
{% csrf_token %}
<table>
{{ form.as_table }}
</table>
<input type="submit" name="Submit">
</form>
{% endblock %}
```

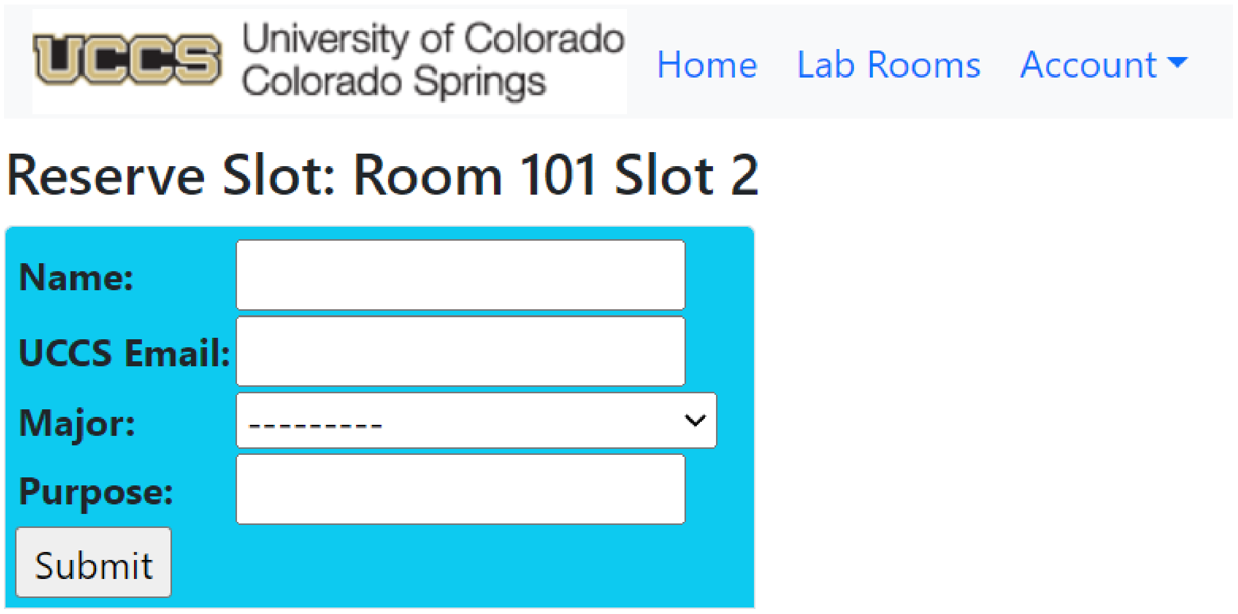
update_slot.html

```
{% block content %}
<h4> Reserve Slot: {{ roomslot }} </h4>
<form action="" method="POST">
{% csrf_token %}
<table>
{{ form.as_table }}
</table>
<a class="btn btn-primary" href="{%url 'labroom-detail' pk %}"
role="button">Cancel</a>
<input type="submit" name="Submit">
</form>
{% endblock %}
```

cancel_slot.html

```
{% block content %}
<h6>Are you sure you want to cancel reservation for
{{roomslot}}?</h6>
<form action="" method="POST">
{% csrf_token %}
<a class="btn btn-primary" href="{%url 'labroom-detail' pk %}"
role="button">Cancel</a>
<input type="submit" value="Submit" />
</form>
{% endblock %}
```

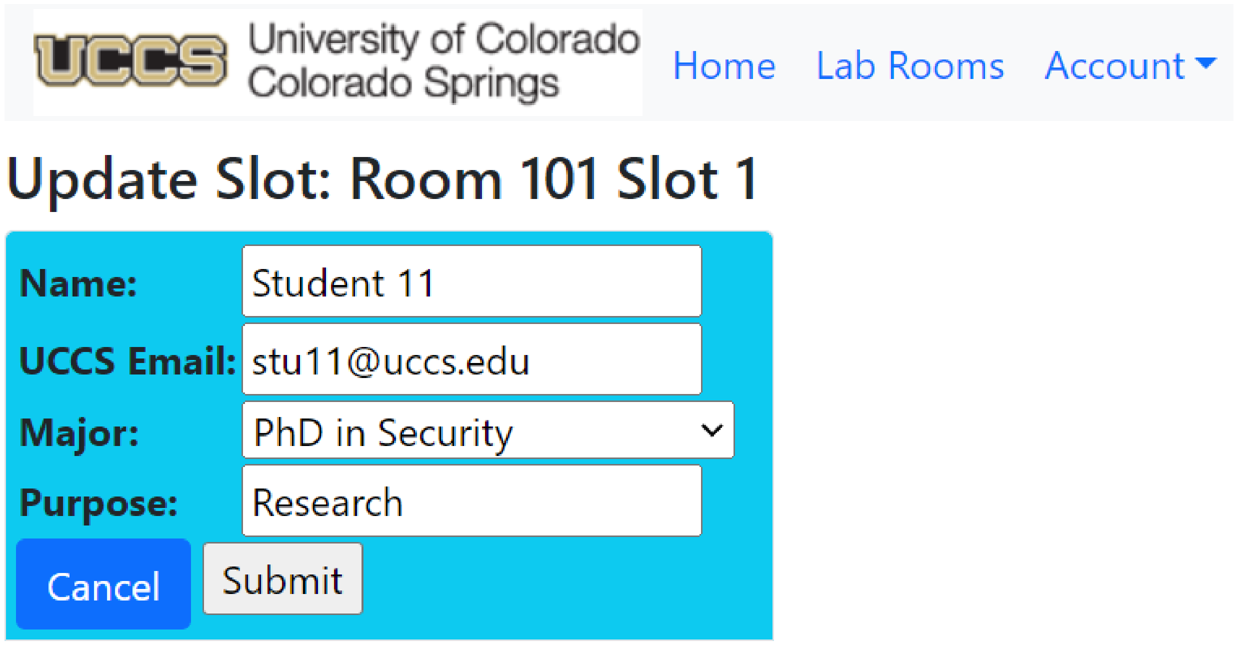
Reserver_slot.html webpage view



Reserve Slot: Room 101 Slot 2

Name:	
UCCS Email:	
Major:	---------
Purpose:	

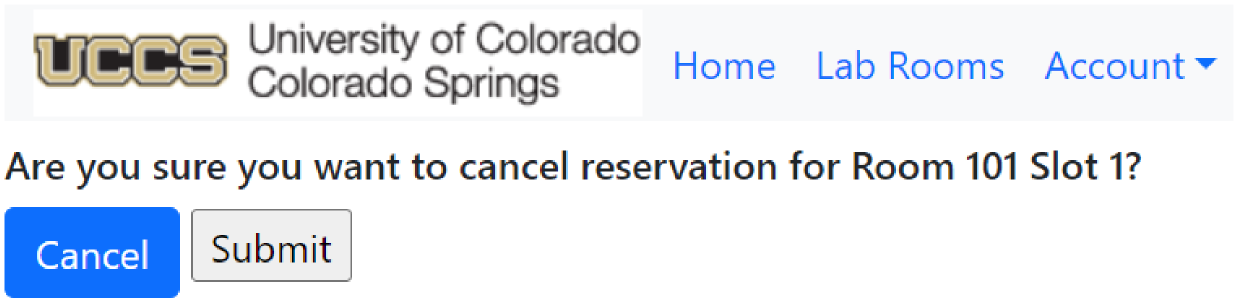
Update_slot.html webpage view



Update Slot: Room 101 Slot 1

Name:	Student 11
UCCS Email:	stu11@uccs.edu
Major:	PhD in Security
Purpose:	Research

Cancel_slot.html webpage view



Are you sure you want to cancel reservation for Room 101 Slot 1?

3.4 Urls

Urls are defined in urls.py file under the application directory. For the sprint 01 following urls are defined:

```
# Index/Homepage
path('', views.index, name='index'),
# List view of the Slots
path('labroom/', views.LabRoomListView.as_view(), name=
'labroom'),
# Detail view of the slot
path('labroom/<int:pk>', views.LabRoomDetailView.as_view(),
name='labroom-detail'),
# Slot reservation
path('labroom/<int:pk>/reserve_slot/', views.reserveSlot,
name='reserve_slot'),
# Canceling the slot
path('labroom/<int:pk>/cancel_slot/', views.cancelSlot,
name='cancel_slot'),
# Updating the slot
path('labroom/<int:pk>/update_slot/', views.updateSlot,
name='update_slot'),
```

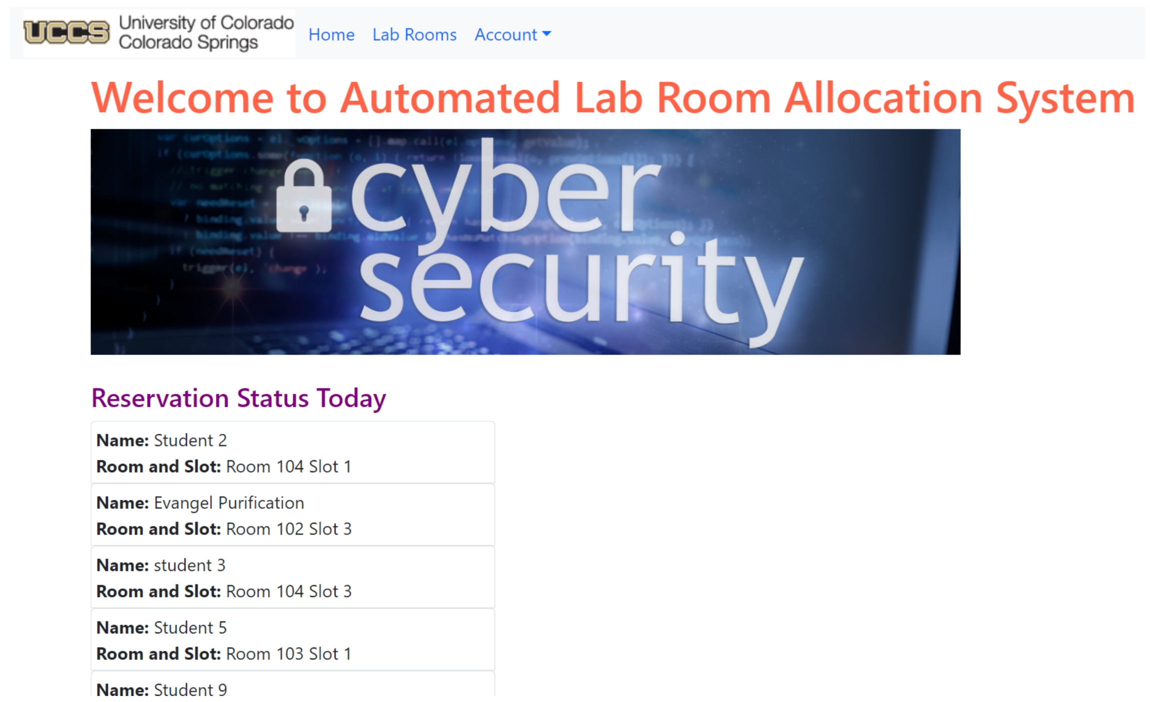
3.5 Responsive Design and Accessibility

In this Sprint01 I used bootstrap for responsive website and accessibility.
I used different font-sizes, text-color and background color for ease of readability and responsive design for wide accessibility.

I use the container class inside <div> tag for formatting a collection of other elements. For separate different Items I use col-sm-4 class. Below is the sample code used for index.html

```
<div class="container justify-content-center">
<h1 style="color:tomato;">Welcome to Automated Lab Room
Allocation System</h1>
<img src="static/images/cyberLab_r.jpeg" class="img-fluid"
alt="Cinque Terre" width="800" height="600">
</div>
<br>
<div class="container justify-content-center">
<h4 style="color:purple;">Reservation Status Today</h4>
{% for i in student_active_today %}
<div class="col-sm-4 p-1 border rounded-top">
    <div class="container-fluid-sm">
    <strong>Name:</strong> {{i.name}}
    </br>
    <strong>Room and Slot: </strong>{{ i.slot.title }}
    </div>
</div>
```

Index.html normal view



Welcome to Automated Lab Room Allocation System

Reservation Status Today

Name: Student 2
Room and Slot: Room 104 Slot 1

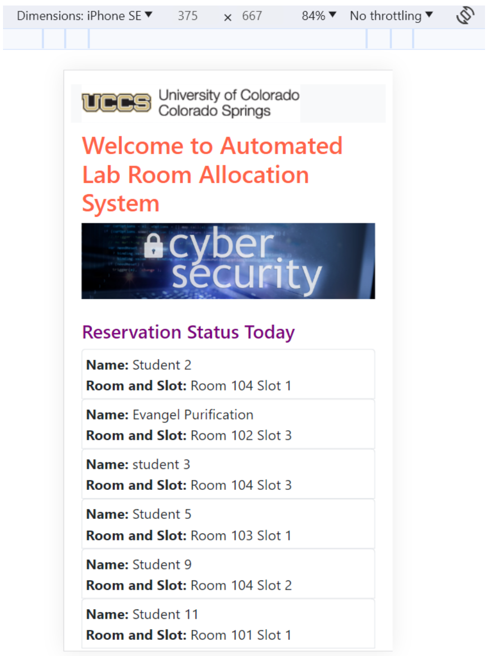
Name: Evangel Purification
Room and Slot: Room 102 Slot 3

Name: student 3
Room and Slot: Room 104 Slot 3

Name: Student 5
Room and Slot: Room 103 Slot 1

Name: Student 9

Index.html responsive view


Dimensions: iPhone SE 375 x 667 84% No throttling
University of Colorado Colorado Springs
cyber security

Welcome to Automated Lab Room Allocation System

Reservation Status Today

Name: Student 2
Room and Slot: Room 104 Slot 1

Name: Evangel Purification
Room and Slot: Room 102 Slot 3

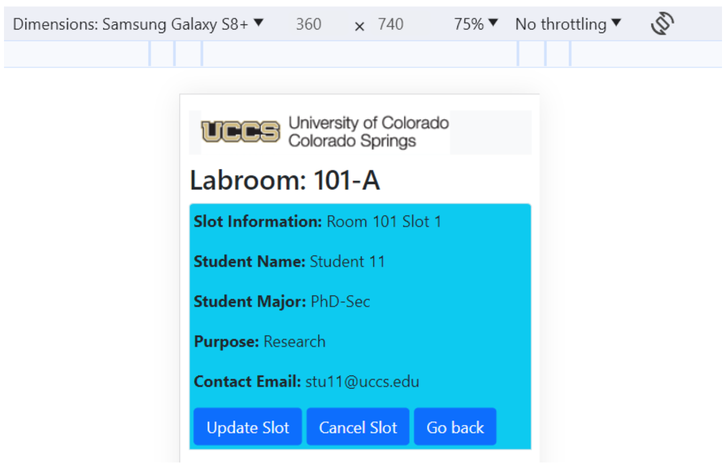
Name: student 3
Room and Slot: Room 104 Slot 3

Name: Student 5
Room and Slot: Room 103 Slot 1

Name: Student 9
Room and Slot: Room 104 Slot 2

Name: Student 11
Room and Slot: Room 101 Slot 1

labroom_detail.html


Dimensions: Samsung Galaxy S8+
360 x 740
75%
No throttling
University of Colorado Colorado Springs

Labroom: 101-A

Slot Information: Room 101 Slot 1

Student Name: Student 11

Student Major: PhD-Sec

Purpose: Research

Contact Email: stu11@uccs.edu

Update Slot
Cancel Slot
Go back

3.6 Resources

Guided Exploration	Resources
GE01	https://docs.google.com/document/d/1Ol0hfri58DhkvgfZYWEIx-8rc6uxRl-A/edit?usp=sharing&ouid=116513778163876913110&rtpof=true&sd=true
GE02	https://docs.google.com/document/d/1nBp3iG05SgYPa4fAdf_zyc_9eChZWZsl/edit?usp=sharing&ouid=116513778163876913110&rtpof=true&sd=true
GE03	https://docs.google.com/document/d/1JPKYAwFG8HIzZCqNqv44LBMdqETnU-oN/edit?usp=sharing&ouid=116513778163876913110&rtpof=true&sd=true
GE04	https://docs.google.com/document/d/1oEFfSHYEZRCxsEz2-9LIkZ51UFZ8di8x/edit?usp=sharing&ouid=116513778163876913110&rtpof=true&sd=true
GE05	https://docs.google.com/document/d/1I-Plw5YuOaaW1YsQEW18KdcE81XJWLSVscdR59Lu3Wo/edit?usp=sharing