BACKEND

```
from flask import Flask, session, render_template, flash, redirect, request
from .models import db, Tenant
from werkzeug.exceptions import BadRequest, InternalServerError

app = Flask(__name__, static_url_path='/static')

username = 'postgres' # default postgres username
password = '1234' # created by the user when they install postgres
host = 'localhost' # host - for local development this is localhost or 127.0.0.1
port = '5432' # default postgres port is 5432
database_name = 'potato_properties'

app.config['SQLALCHEMY_DATABASE_URI'] = f'postgresql://{username}:{password}@{host}:{port}/{database_name}'
# Session uses this key to cryptographically sign cookies
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'
db.init_app(app)

with app.app_context():
    db.create_all()
```

These lines of code is the initialization of the web application. This application uses Flask as a backend framework. A postgresql database is also initialized with the flask application as can be seen in lines 8-17 ending with db.init_app(app). The last few lines take the SQLAlchemy ORM models and construct the postgres tables that are used within the application. The app.secret_key on line 16 is used for sessions. This application uses session authentication to allow for users to login and logout. The secret key is needed to sign a cookie which holds all the session data, which includes the user email.

```
@app.get('/')
def landing():
    if 'email' in session:
        return redirect('/dashboard')

    return render_template('index.html')

@app.get('/signup')
def get_signup():

    if 'email' in session:
        return redirect('/dashboard')

    return render_template('signup.html')
```

These two routes are for the login and signup pages. Both routes first check if there is an email attribute in the global session variable. The reason for this is when the user signs in, an email attribute is added to the session object in which a cookie is signed. This means that if there is no email attribute in session, then there is no cookie implying a user is not signed in. A user should not be able to see the login or signup page if they are currently logged in.

```python
@app.post('/api/create-user')
def create_user():

    name = request.form.get('name')
    email = request.form.get('email')
    password = request.form.get('password')
    confirm_password = request.form.get('confirm-password')

    if name == '' or email == '' or password == '' or confirm_password ==
'':
        raise BadRequest('Invalid Credentials: Please fill out all
fields.')

    if password != confirm_password:
        flash('Passwords do not match!', 'error')
        return redirect('/signup')

    potential_user = Tenant.query.filter_by(email=email).first()

    if potential_user:
        flash('A user with that email already exists!', 'error')
        return redirect('/signup')

    user = Tenant(name=name, email=email, password=password)

    db.session.add(user)
    db.session.commit()

    flash('Account has been created!', 'info')
    return redirect('/')
```

This route is for creating a user. As we can see, this route is called upon submission from the signup page. We retrieve the credentials from the submitted form and check for any errors. If there are errors we render them and redirect to the signup page. However, if everything checks out, then a user is constructed and added to the database, then the user is redirected to the login page with an account created message.

```python
@app.post('/')
def login():

    email = request.form.get('email')
    password = request.form.get('password')

    user = Tenant.query.filter_by(email=email, password=password).first()

    if not user:
        flash('Invalid email or password', 'error')
        return redirect('/')

    session['email'] = email

    return redirect('/dashboard')

@app.post('/logout')
def logout():
    del session['email']
    return redirect('/')
```

These routes are for logging into an existing account as well as logging out of the current account. As we can see with the login route, we retrieve the form data and filter the database looking for an existing user. If a tenant is found with the given credentials, then the user email is added to the session object and the user is redirected to the dashboard. However, if the user is not found, then the user is redirected to the homepage with an error message saying invalid email or password.

For the logout route, the user is assumed to be logged in as this route can only be accessed via a button click on a logged in only route. This route removes the email attribute from the session attribute and redirects the user to the login page.

```python
@app.get('/dashboard')
def get_dashboard():
    if 'email' not in session:
        return redirect('/')

    email = session['email']

    user = Tenant.query.filter_by(email=email).first()

    if not user:
        raise InternalServerError('Something went wrong.')

    return render_template('dashboard.html', user=user)

@app.get('/maintenance')
def get_maintenance():
    if 'email' not in session:
        return redirect('/')

    return render_template('maintenance.html')
```

These two routes are for logged in users only. The dashboard route is the main route where the user can see all their information. It first checks if email is in the session, for if it is not, then the user is redirected to the login page. If the email is in the session object, then it can be assumed the user is logged in and we use the email to search the database for the user. Assuming the user is found, the dashboard page is rendered with the user object in which the user object information can be seen in the HTML. This is achieved with server side rendering using the Jinja2 templating engine that Flask web framework provides.

```
@app.errorhandler(404)
def not_found(error):
    return render_template('error.html', message=error), 404

@app.errorhandler(400)
def bad_request(error):
    return render_template('error.html', message=error), 400

@app.errorhandler(500)
def server_error(error):
    return render_template('error.html', message=error), 400
```

Lastly, these routes provide error handling routes for any errors that may arise. The 404 error handler route of course handles any invalid routes that a user may enter into the search bar. In this event, the user is greeted with a not found error message. The 400 error handler route provides handling for any invalid input that a user may enter. The forms on each of the pages provide client-side validation to try and prevent bad input data, however, if a user tries to get around this by editing the HTML, they will still be greeted with an error screen due to this route. The 500 error handler route is for any other error that is not accounted for. This one will be triggered in an unlikely event a database search takes place and there is no current user set.

```python
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import TIMESTAMP

db = SQLAlchemy()

class Tenant(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=False)
    email = db.Column(db.String, unique=True, nullable=False)
    password = db.Column(db.String, nullable=False)

    def __repr__(self) -> str:
        return f'Tenant: {self.name}, {self.email}'
```

This functionality can be seen in the models.py file located in the src directory. This file provides a python class which serves as a bridge to accessing the underlying postgresql database. This is achieved with the SQLAlchemy dependency. We can query, add, and remove data from the database all through using this Tenant class.

FRONTEND

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://cdn.tailwindcss.com"></script>
    <link rel="stylesheet" href="/static/css/style.css" /> <!-- Tailwind
-->
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.6.0/css/all.mi
n.css"
integrity="sha512-Kc323vGBEqzTmouAECnVceyQqyqdsSiqLQISBL29aUW4U/M7pSPA/gEU
ZQqv1cwx4OnYxTxve5UMg5GT6L4JJg==" crossorigin="anonymous"
referrerpolicy="no-referrer" /> <!-- FontAwesome (icons) -->
    {% block title %}{% endblock %}
</head>
<body class="relative">
    {% if request.path == '/' or request.path == '/signup' %}
        <div class="header">
            <a href="/"><img src="/static/images/potato_logo.png"
alt="Potato Properties Logo"></a>
        </div>
    {% endif %}
    {% block content %}{% endblock %}
    <script src="../static/js/index.js" defer></script>
</body>
</html>
```

This file serves as the layout template for all the other templates we have. As was mentioned this is a template file, which can be made more obvious when we see the {% %} syntax located in some of the lines.This file includes our stylesheets, Tailwind CSS import, and FontAwesome icon imports which can then be used in all our other frontend templates that are injected into this layout file.

```
{% extends '_layout.html' %}
{% block title %}<title>PotatoProperties | Login</title>{% endblock %}
{% block content %}
<div class="login-container">
    {% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %}
        {% for category, message in messages %}
        <div id="message-container" class="{{ category }} flex
justify-center">
            <div class="flex-1"></div>
            <h1 class="flex-1">{{ message }}</h1>
            <div class="flex-1 flex justify-end"><button
id="flash-message">&times;</button></div>
        </div>
        {% endfor %}
    {% endif %}
    {% endwith %}
    <form action="/" method="post">
        <div class="input-group">
        <label for="email">Email</label>
        <input type="email" id="email" name="email" placeholder="Enter
Your Email" required>
        </div>
        <div class="input-group">
        <label for="password">Password</label>
        <input type="password" id="password" name="password"
placeholder="Enter Your Password" required>
        </div>
        <div class="button-group">
            <button type="submit">Log In</button>
            <a href="/signup"><button type="button" class="signup
w-full">Sign Up</button></a>
        </div>
    </form>
</div>
{% endblock %}
```

This file serves as the login page. This file provides “blocks” which are identifiable by the name next to the keyword “block”. These blocks are injected into the layout file when these HTML files are called with the render_template function on the backend. We know this file will be injected into _layout.html with the {% extends '_layout.html' %} tag at the top.

```
{% extends '_layout.html' %}

{% block title %}<title>PotatoProperties - Sign up</title>{% endblock %}

{% block content %}
<div class="signup-container">
    {% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %}
        {% for category, message in messages %}
        <div id="message-container" class="{{ category }} flex
justify-center">
            <div class="flex-1"></div>
            <h1 class="flex-1">{{ message }}</h1>
            <div class="flex-1 flex justify-end"><button
id="flash-message">&times;</button></div>
        </div>
        {% endfor %}
    {% endif %}
    {% endwith %}
    <form action="/api/create-user" method="post">
        <div class="input-group">
            <label for="name">Full Name</label>
            <input type="text" id="name" name="name" placeholder="Enter
Your Full Name" required>
        </div>
        <div class="input-group">
            <label for="email">Email</label>
            <input type="email" id="email" name="email" placeholder="Enter
Your Email" required>
        </div>
        <div class="input-group">
            <label for="password">Password</label>
            <input type="password" id="password" name="password"
placeholder="Enter Your Password" required>
        </div>
        <div class="input-group">
            <label for="confirm-password">Confirm Password</label>
            <input type="password" id="confirm-password"
name="confirm-password" placeholder="Confirm Your Password" required>
        </div>
        <div class="button-group">
            <button type="submit">Sign Up</button>
            <a href="/"><button type="button" class="w-full">Log
        In</button></a>
        </div>
    </form>
</div>
{% endblock %}
```

Just like with index.html (login page file), this file uses the same block names to inject HTML code into the _layout.html file.

```
{% extends '_layout.html' %}

{% block title %}<title>PotatoProperties | Dashboard</title>{% endblock %}
{% block content %}
<div class="h-full w-full flex">
    <div class="sidebar">
        <a href="/dashboard"><img src="/static/images/potato_logo.png"
alt="Potato Properties Logo"></a>
        <h2 style="color: #443D34;">Unit 002-A</h2>
        <a href="/dashboard" class="menu-item active">Dashboard</a>
        <a href="/maintenance" class="menu-item">Maintenance</a>
        <a href="#" class="menu-item">Contact Us</a>
        <br>
        <br>
        <br>
        <form action="/logout" method="post" class="text-left w-full
text-white text-[20px] font-bold p-[10px] mr-[25px] rounded-[4px]
bg-[#77654fe0]"><button class="w-full text-left">Logout</button></form>
    </div>
    <div class="dashboard-content">
        <div id="dash-head" style="margin-top: 50px;">
            <h1 style="font-size: 35px;">Dashboard</h1>
            <div class="profile-icon">
                <i class="fa fa-user"></i>
            </div>
        </div>
        <br>
        <h2 style="color: #443D34; font-size: 40px;">Welcome back, {{
user.name }}!</h2>
        <div class="balance-section">
            <div>
                <p class="balance-amount" style="color: #443D34;
font-size: 50px;
                margin-bottom: 0px;">$0.00</p>
                <p style="color: #443D34; font-size: 25px; margin-bottom:
2px;"><b>No Balance Due</b></p>
            </div>
            <button class="balance-button" style="margin-top: 80px;
background-color:
            #948A76; color: #443D34; font-size: 20px;"><b>Make A
Payment</b></button>
        </div>
        <div class="news-section">
            <h2 style="color: #443D34; font-size: 30px;">News &
Updates</h2>
            <div class="news-item"></div>
            <div class="news-item"></div>
            <div class="news-item"></div>
        </div>
        <h2 style="color: #443D34; font-size: 30px;">Current Lease</h2>
        <div class="scrollable-section">
            <div class="lease-info">
                <p><b>Lease Start Date:</b> January 1, 2024</p>
                <p><b>Lease End Date:</b> December 31, 2024</p>
                <p><b>Monthly Rent:</b> $1,200</p>
                <img src="{{ url_for('static', filename='lease.png') }}"
alt="Lease Agreement">
            </div>
        </div>
        <h2 style="color: #443D34; font-size: 30px;">Payment History</h2>
        <div class="scrollable-section">
            <div class="payment-history">
                <img src="{{ url_for('static',
filename='payment_history.png') }}"
                alt="Payment History">
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

This file is similar to the last two HTML templating files where the blocks are injected into _layout.html. However, there is one key difference. This file takes a user object from the backend and injects this user data into the HTML, which is then injected into the _layout file. We can see this dynamic user data seen in the line that is highlighted white. This line allows for the HTML to show up differently depending on who is logged in.

```
{% extends '_layout.html' %}

{% block title %}<title>PotatoProperties | Maintenance</title>{% endblock
%}
{% block content %}
<div class="h-full w-full flex">
    <div class="sidebar">
        <a href="/dashboard"><img src="/static/images/potato_logo.png"
alt="Potato Properties Logo"></a>
        <h2 style="color: #443D34;">Unit 002-A</h2>
        <a href="/dashboard" class="menu-item">Dashboard</a>
        <a href="/maintenance" class="menu-item active">Maintenance</a>
        <a href="#" class="menu-item">Contact Us</a>
        <br>
        <br>
        <br>
        <form action="/logout" method="post" class="text-left w-full
text-white text-[20px] font-bold p-[10px] mr-[25px] rounded-[4px]
bg-[#77654fe0]"><button class="w-full text-left">Logout</button></form>
    </div>
    <div>
        <h1>Maintenance</h1>
    </div>
</div>
{% endblock %}
```

Next we have the maintenance file which is just like the dashboard page. It is again injected into _layout.html.

```
{% extends '_layout.html' %}

{% block title %}<title>PotatoProperties | Error</title>{% endblock %}

{% block content %}
<div class="header">
    <a href="/"><img src="/static/images/potato_logo.png" alt="Potato
Properties Logo"></a>
</div>
<h1 class="font-bold">{{message}}</h1>
{% endblock %}
```

Lastly, we have the error page which is again injected into _layout.html. This error page serves as the template for all types of errors. The message from the error handler function is passed into the template which is then displayed in the center of the screen.