Capstone Project Documentation: Paws Home Volunteer Management System

Group 7: Yating Liu, Haoguo Cheng, Richard Liu

School of Professional Studies, Northwestern University

CIS 498-0-50: Information Systems Capstone

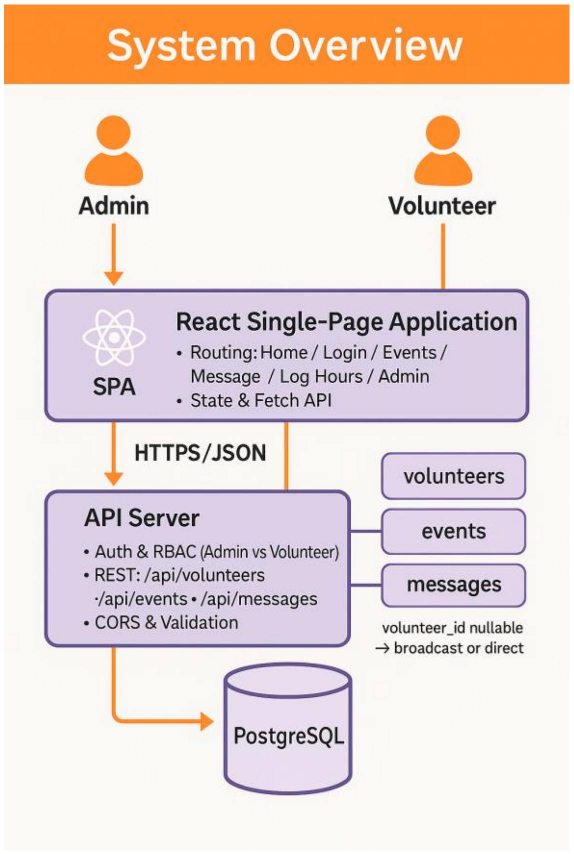
Instructor: Amul Chapla

August 28, 2025

Content

1 Production Support Document & Testing Scenarios

1.1 Service dependency diagram


System Overview
Admin
Volunteer
React Single-Page Application
• Routing: Home / Login / Events / Message / Log Hours / Admin
• State & Fetch API
SPA
HTTPS/JSON
API Server
• Auth & RBAC (Admin vs Volunteer)
• REST: /api/volunteers ·/api/events • /api/messages
• CORS & Validation
volunteers
events
messages
volunteer_id nullable → broadcast or direct
PostgreSQL

For the cloud deployment, this application use **Railway** for hosting frontend, backend and database.

1.2 Monitoring

1.2.1 Frontend

Logs Location: Logs are available on the Railway dashboard under the Frontend service → Logs.

Monitoring Tasks:

- Page load failures: Look for HTTP status codes 404, 500.
- API call failures: Network errors in DevTools → check if API URL is misconfigured.
- UI performance issues: Long rendering times, unresponsive buttons, broken layout (especially on mobile).

Health Checks and Alerts: Implement a lightweight endpoint in the frontend like /health-check that returns a small HTML or JSON payload ({"status":"ok"}).

1.2.2 Backend

Logs Location: Railway dashboard → Backend service → **Logs** tab.

Monitoring Tasks:

- API errors: 500 Internal Server Error, 400 Bad Request.
- Authentication issues: Failed login attempts, JWT token errors.
- Database connection errors: psycopg2.OperationalError.
- Resource usage: Memory, CPU spikes (check Railway metrics).

Health Check Endpoint:

```python
@app.route("/api/health")
def health():
    return {"status": "ok"}, 200
```

Call this endpoint periodically (every 1-5 minutes) using a monitoring tool. If not returning 200, trigger alert.

Automated Monitoring Recommendations:

- Configure Railway alerts for service crashes.
- Add logging for all API requests and errors.
- Optional: Send error logs to a central logging service like Papertrail.

1.2.3 Database

Metrics Location: Railway Database → Metrics tab shows connection count, queries per second, CPU/memory usage. Errors like duplicate key or connection refused appear in backend logs.

Monitoring Tasks:

- Monitor connection pool size: Ensure no exhausted connections.
- Monitor slow queries: Use PostgreSQL logs or Railway metrics.
- Backup status: Verify automated backups in Railway settings.

Alerting: If DB downtime or query failures occur, backend health check will fail. Optionally, set Railway alerts or email notifications when DB becomes unavailable.

1.3 Common Incidents & Recovery Steps

1.3.1 Database Connection Loss

Backend logs show “psycopg2.OperationalError: could not connect to server”.

Recovery Steps:

(1) Verify Railway database is running.

(2) Check backend config.py to ensure SQLALCHEMY_DATABASE_URI is correct.

(3) Restart backend service in Railway.

1.3.2 Backend Service Crash

Railway backend logs show “RuntimeError” or “500 Internal Server Error”.

Recovery Steps:

(1) Check logs in Railway for Python errors.

(2) Run flask run locally with same config to reproduce.

(3) Fix code → redeploy → verify.

1.3.3 Frontend Not Loading / 404 Errors

Browser shows blank page or broken UI.

Recovery Steps:

(1) Check Railway frontend logs to confirm build success.

(2) Verify REACT_APP_API_URL points to backend service URL.

(3) Clear browser cache and then reload.

1.4 Testing scenarios & result

Purpose

- Before deploy: prove correctness, detect regressions early.
- After deploy: quickly validate the system is healthy and the release “took”.
- During incidents: reproduce and verify fixes with repeatable checks.

1.4.1 Unit tests

For example, testing API Routes:

POST /api/register creates user.

POST /api/login returns JWT token.

POST /api/events/:id/register enforces unique constraint.

1.4.2 Integration tests

Verify route/controller ↔ service ↔ DB together

1.4.3 End-to-end (E2E)

Scenario 1:

- User signs up → logs in → registers for event.
- Expected: Event registration success.
- Actual: Pass.

Scenario 2:

- User tries full event registration.
- Expected: Event can not be registered.

- Actual: Pass.

1.4.4 Manual tests

Test Case	Action	Expected	Actual Status
User Registration	Fill signup form	Success → redirect to schedule	Works
Login	Enter correct credentials	Dashboard loads	Works
Book a Shift	Select the date and time slot	Added to schedule list	Works
Wrong Password	Enter invalid password	Error message	Works
Event Registration	Click "Join Event"	Added to event list	Works
Log volunteer hours	Fill the log hours form	Added the record	Works

1.4.5 Smoke tests (post-deploy)

Run immediately after deployment:

1. Visit frontend URL → page loads.
2. Register new user → confirm in DB.
3. Login with same user → returns token.
4. Register event → confirm record in DB.

2 System Setup Instructions (Frontend, Backend, Database)

2.1 Prerequisites

2.1.1 Operating System

Windows 10/11, macOS, or Linux

2.1.2 Runtime and Packages

(1) Frontend (React)

Runtime Version:

- Node.js: v20.11.1
- npm: 10.2.4

Key Packages:

Library Name	Version	Description
react	^19.1.0	Core React library for building UI components
react-dom	^19.1.0	DOM-specific methods for rendering React apps
react-scripts	5.0.1	Scripts and configuration for Create React App (CRA)
react-router-dom	^6.30.1	Routing library for handling navigation in React apps
web-vitals	^2.1.4	Library for measuring and reporting web performance metrics
@testing-library/react	^16.3.0	Utilities for testing React components
@testing-library/jest-dom	^6.6.3	Custom DOM element matchers for Jest
@testing-library/dom	^10.4.0	Low-level DOM testing utilities
@testing-library/user-event	^13.5.0	Simulates user interactions (click, type, etc.) in tests

(2) Backend (Flask)

Runtime Version:

- Python: 3.10.x

Key Packages:

Library Name	Version	Description
Flask	2.2.2	Web Framework
Flask-SQLAlchemy	2.5.1	ORM Integration for Flask
SQLAlchemy	1.4.48	Database ORM Library

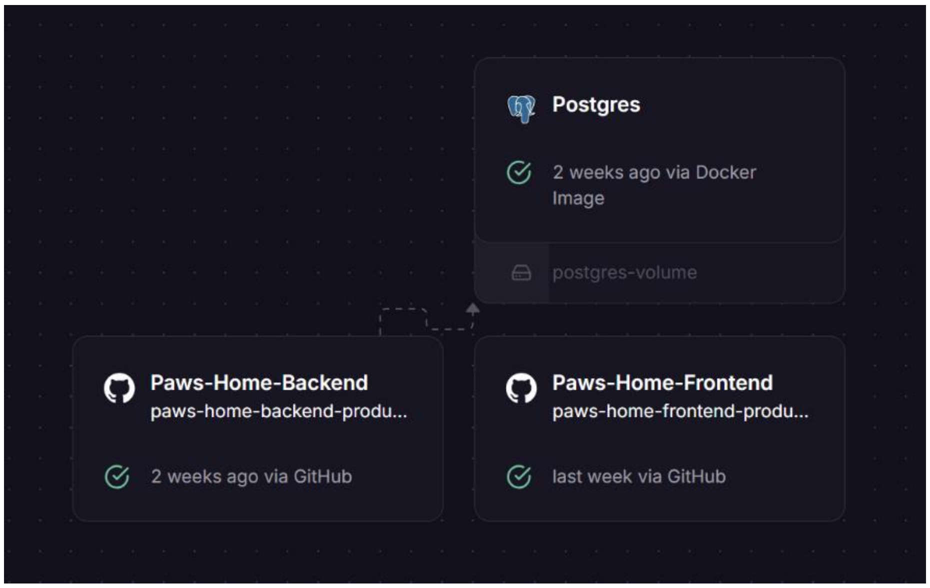
Library Name	Version	Description
Flask-Migrate	3.1.0	Database Migration Tool
Flask-Cors	3.0.10	Cross-Origin Resource Sharing Support
python-dotenv	0.19.2	Environment Variable Management
gunicorn	21.2.0	Production WSGI Server for Deployment
psycopg2-binary	2.9.7	PostgreSQL Database Adapter
Werkzeug	2.2.3	Utility library for Flask, WSGI toolkit

(3) Database (PostSQL)

Runtime Version: 11.x

2.1.3 Cloud Services

Railway for hosting frontend, backend and database.



2.1.4 Environment Variables

DATABASE_URL: PostgreSQL connection string

REACT_APP_API_URL: Backend API endpoint

2.2 Installation Steps (localhost)

2.2.1 Frontend

Step 1: Clone the repository

```
git clone https://github.com/Yating0521/Paws-Home-Frontend
cd paws-home-frontend
```

Step 2: Install dependencies

```
npm install
```

Step 3: Configure environment variables
In the root folder, create a .env file:

```
REACT_APP_API_URL=http://localhost:5000
```

Step 4: Run frontend locally

```
npm start
```

Access app at http://localhost:3000

2.2.2 Backend

Step 1: Clone the repository

```
git clone https://github.com/Yating0521/Paws-Home-Backend
cd paws-home-backend
```

Step 2: Create a virtual environment

```
python -m venv venv
source venv/bin/activate      # macOS/Linux
venv\Scripts\activate         # Windows
```

Step 3: Install dependencies

```
pip install -r requirements.txt
```

Step 4: Initialize the database
In app/config.py, ensure your SQLALCHEMY_DATABASE_URI is correct:

```
SQLALCHEMY_DATABASE_URI =
"postgresql://postgres:<password>@localhost/paws_home_vms"
```

Step5: Run backend locally

flask run

Access API at http://localhost:5000.

2.2.3 Database

Step 1: Install PostgreSQL (skip if Railway provides DB).

Step 2: Create a new database

```sql
CREATE DATABASE paws_home_vms;
```

Step 3: Update backend config with DB credentials (local or Railway).

2.3 Configuration Details

(1) **config.py** holds database connection & Flask settings.

(2) Use **.env** for secrets in Railway. Example:

.env file contains API URL (REACT_APP_API_URL).

(3) Railway Deployment:

Set environment variables under Project → Variables.

Procfile example (backend):

web: gunicorn run:app

2.4 Build and Deployment Steps

2.4.1 Backend Deployment (Railway)

(1) Push code to GitHub.

(2) Connect GitHub repo to Railway project.

(3) Configure environment variables on Railway.

(4) Deploy → Railway builds and serves Flask app.

2.4.2 Frontend Deployment (Railway)

(1) Push frontend repo to GitHub.

(2) Connect to Railway → Create new service.

(3) Add .env with backend API URL (Railway backend URL).

(4) Deploy → Railway serves React app.

2.4.3 Database (Railway)

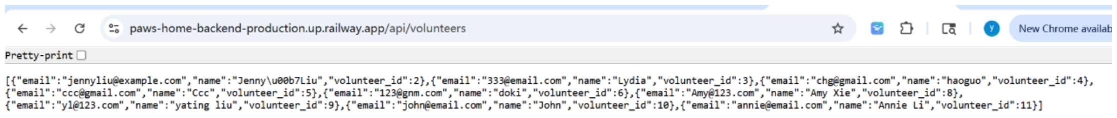
(1) Create PostgreSQL service in Railway.

(2) Copy connection string → set as DATABASE_URL in backend.

2.5 Validation

2.5.1 Backend Test

Visit http://localhost:5000/api/volunteers (or the domain address generated by the Railway, e.g., http://paws-home-frontend-production.up.railway.app/api/volunteers), Should return JSON (empty list if no volunteers). You can test other APIs to ensure the backend functionalities.



paws-home-backend-production.up.railway.app/api/volunteers

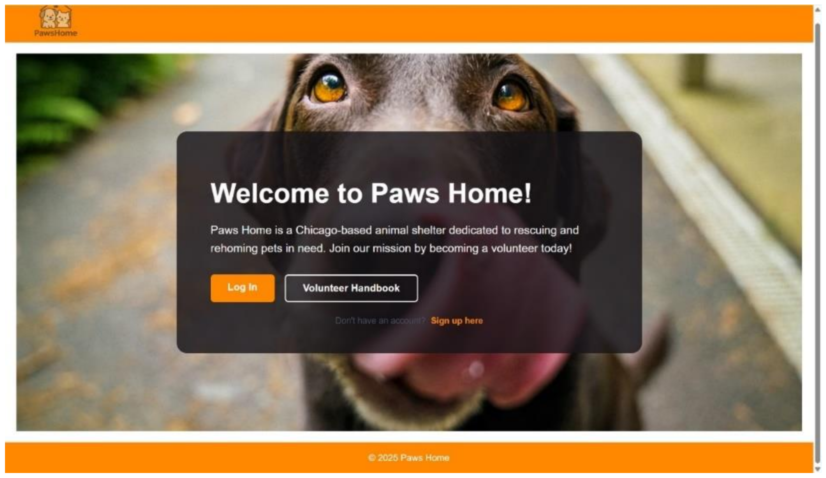
Pretty-print

```
[{"email":"jennyliu@example.com","name":"Jenny\u00b7Liu","volunteer_id":2},{"email":"333@email.com","name":"Lydia","volunteer_id":3},{"email":"chg@gmail.com","name":"haoguo","volunteer_id":4},
{"email":"ccc@gmail.com","name":"Ccc","volunteer_id":5},{"email":"123@gnm.com","name":"doki","volunteer_id":6},{"email":"Amy@123.com","name":"Amy Xie","volunteer_id":8},
{"email":"yl@123.com","name":"yating liu","volunteer_id":9},{"email":"john@email.com","name":"John","volunteer_id":10},{"email":"annie@email.com","name":"Annie Li","volunteer_id":11}]
```

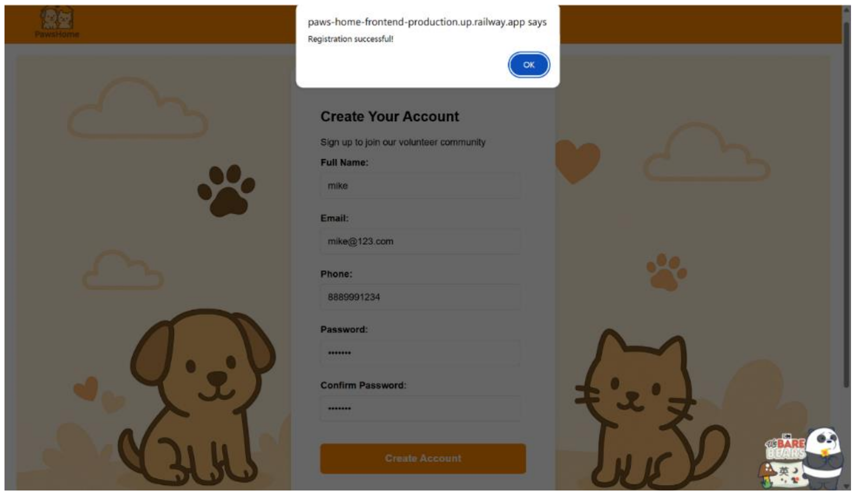
2.5.2 Frontend and Database Test

Visit http://localhost:3000 (or the domain address generated by the Railway, e.g., http://paws-home-frontend-production.up.railway.app) to see if the Homepage loads. After that, register as a volunteer and confirm data appears in DB.

Homepage:


PawsHome
Welcome to Paws Home!
Paws Home is a Chicago-based animal shelter dedicated to rescuing and rehoming pets in need. Join our mission by becoming a volunteer today!
Log In
Volunteer Handbook
Don't have an account? Sign up here
© 2025 Paws Home

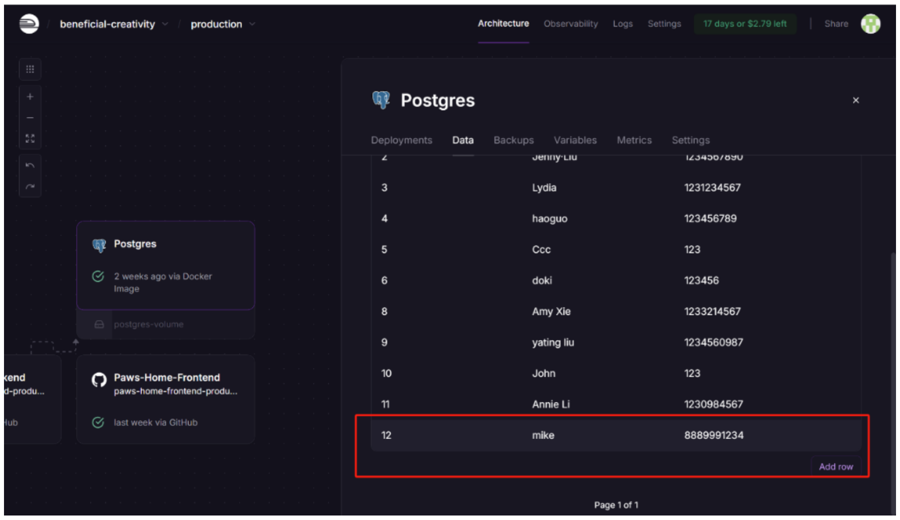
Registration:



paws-home-frontend-production.up.railway.app says
Registration successful!
OK

Create Your Account
Sign up to join our volunteer community
Full Name: mike
Email: mike@123.com
Phone: 8889991234
Password: •••••••
Confirm Password: •••••••
Create Account

Confirm in the database:



2	Jenny-Liu	1234567890
3	Lydia	1231234567
4	haoguo	123456789
5	Ccc	123
6	doki	123456
8	Amy Xie	1233214567
9	yating liu	1234560987
10	John	123
11	Annie Li	1230984567
12	mike	8889991234

2.5.3 Deployed App Test (Railway)

Visit Railway frontend URL and try registering and signing in.

3 Issue Diagnosis, Research, Resolution, and Sharing

3.1 Issue 1: Frontend Could Not Connect to Backend on Railway

(1) Issue Description
Frontend React app deployed on Railway could not fetch API data.

Expected: frontend connects to backend API.

Actual: requests failed with CORS and Network Error.

(2) Environment & Setup Details

- Backend deployed with Procfile on Railway
- Frontend deployed separately on Railway
- PostgreSQL database hosted on Railway

(3) Steps to Reproduce

Step 1: Deploy backend on Railway.

Step 2: Deploy frontend with API URL configured.

Step 3: Try login/register but failed with network error.

(4) Diagnosis:

- Backend was sleeping until accessed directly.
- REACT_APP_API_URL was not correctly set in frontend .env.
- CORS headers missing in Flask backend.

(5) Research Process

Reviewed Railway deployment logs and checked Flask-CORS documentation. Asked ChatGPT for debugging API connectivity.

(6) Resolution Steps

Step 1: Installed Flask-CORS and enabled:

```python
from flask_cors import CORS
CORS(app)
```

Step 2: Updated frontend .env

```
REACT_APP_API_URL=https://<backend-service-url>.railway.app
```

Step 3: Redeployed frontend.

(7) Outcome Verification

- Confirmed requests to backend succeeded.
- Logs showed 200 OK responses from API.
- User registration and login worked.

3.2 Issue 2: Empty Body

(1) Problem Description

On the AdminVolunteer page, after composing a message and clicking Send, the UI showed an error (or silently failed). The browser console displayed:

SyntaxError: Unexpected end of JSON input

POST /api/messages returned 201 Created, but the response body was empty (Content-Length: 0).

Expected: After sending, the UI should confirm success and the volunteer should see the new item in Message.

Actual: DB row was created (verified in pgAdmin), but the front end failed while parsing the server's empty response body.

(2) Environment & Setup

- OS / Browser: Windows 11 / Chrome 126
- Frontend: React, runs at http://localhost:3000
- Backend: Flask + Flask-CORS at http://127.0.0.1:5000
- SQLAlchemy connected to PostgreSQL 11

(3) Steps to Reproduce

Step 1: Start Flask backend and React frontend.

Step 2: Open AdminVolunteer page, pick a volunteer → click Message.

Step 3: Fill Title and Content, click Send.

Step 4: Open DevTools → Network → select POST /api/messages: status 201, no response body.

Step 5: Look at Console: *SyntaxError: Unexpected end of JSON input.*

Step 6: Switch to a volunteer account and open Message → new message isn't shown (UI never updated).

(4) Diagnosis

The React code unconditionally executed await r.json() after the POST.

The server returned 201 Created with an empty body. Parsing an empty body as JSON throws a SyntaxError. MDN documents that Response.json() throws if the body "cannot be parsed as JSON," which includes an empty body. [1]

It is valid for 201 responses to omit a message body; the status only indicates a resource was created. (Body is optional per HTTP semantics.) [2]

Therefore, the mismatch was: client assumes JSON vs server sometimes responds with no body → parse error.

(5) Research

[1] MDN (Response.json) — throws SyntaxError when body can't be parsed as JSON (e.g., empty). https://developer.mozilla.org/en-US/docs/Web/API/Response/json

[2] HTTP 201 semantics — resource created; response body is optional.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status/201

https://www.rfc-editor.org/rfc/rfc9110.html

[3] Empty/204 bodies + fetch — widely reported that calling json() on empty bodies triggers “Unexpected end of JSON input.”

https://github.com/whatwg/fetch/issues/113

https://stackoverflow.com/questions/65815485/status-204-response-json-caught-syntaxerror-unexpected-end-of-json-input-a

[4] Flask jsonify — recommended way to return JSON with correct Content-Type.

https://flask.palletsprojects.com/en/stable/api/

[5]Flask-CORS — confirm CORS setup for local dev.

https://flask-cors.readthedocs.io/en/latest/

(6) Resolution Steps

Step 1: Server-side (recommended primary fix)

Return a JSON body for the create endpoint so the client can safely parse it and optimistically update the UI. [4]

```
return jsonify({"message_id": new_message.message_id}), 201
```

Step 2: Client-side robustness (defensive, keep even with A Guard JSON parsing in case some future responses are empty or non-JSON. [1,3]

```
const r = await fetch(`${API_BASE}/api/messages`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ ...msgForm, volunteer_id: messagingId }),
});

if (!r.ok) throw new Error(`HTTP ${r.status}`);

const ct = r.headers.get('content-type') || '';
const data = ct.includes('application/json') ? await r.json() : null;
```

(7) Verification

- API check (curl): HTTP/1.1 201 CREATED and a JSON body containing message_id and fields.

```
curl -i -X POST http://127.0.0.1:5000/api/messages \
  -H "Content-Type: application/json" \
  -d '{"title":"Test","content":"Hi!","sender_name":"Admin","volunteer_id":1}'
```

- UI flow (Admin): After Send, the composer closes, a success toast appears, and no console error is thrown.
- UI flow (Volunteer): Opening Message issues GET /api/messages?volunteer_id=1; the new item appears in the left list; the right panel shows full content.
- Network tab: POST shows 201 with Content-Type: application/json. GET returns the full list (broadcast + directed), ordered by date.
- Database: pgAdmin shows a new row in messages with the expected volunteer_id and date.

4 System Usage Guide

4.1 Accessing the application

Web app URL: https://paws-home-frontend-production.up.railway.app

Supported Devices: Desktop, tablet, and mobile (responsive design applied).

Test Accounts:

(1) Admin

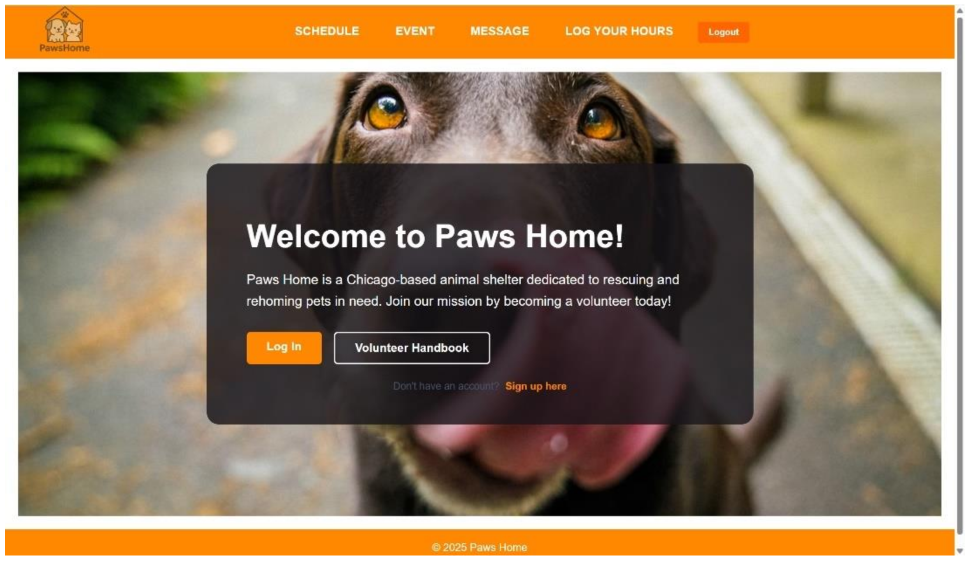
Email: `alice@pawshome.com`

Password: `password123`

(2) Volunteer: Create accounts via the Register page.

4.2 Navigating key features

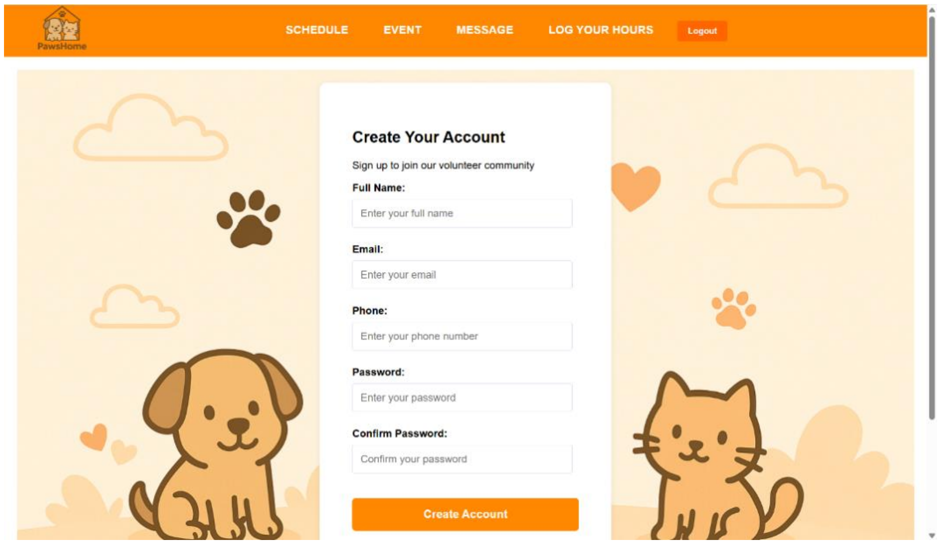
(1) Homepage: Quick links to Log In, Register, and Volunteer Handbook.



Welcome to Paws Home!

Paws Home is a Chicago-based animal shelter dedicated to rescuing and rehoming pets in need. Join our mission by becoming a volunteer today!

(2) Register: Create an account with name, email, phone, and password. (Click sign up here on the homepage or sign in page)



Create Your Account

Sign up to join our volunteer community

Full Name: Enter your full name

Email: Enter your email

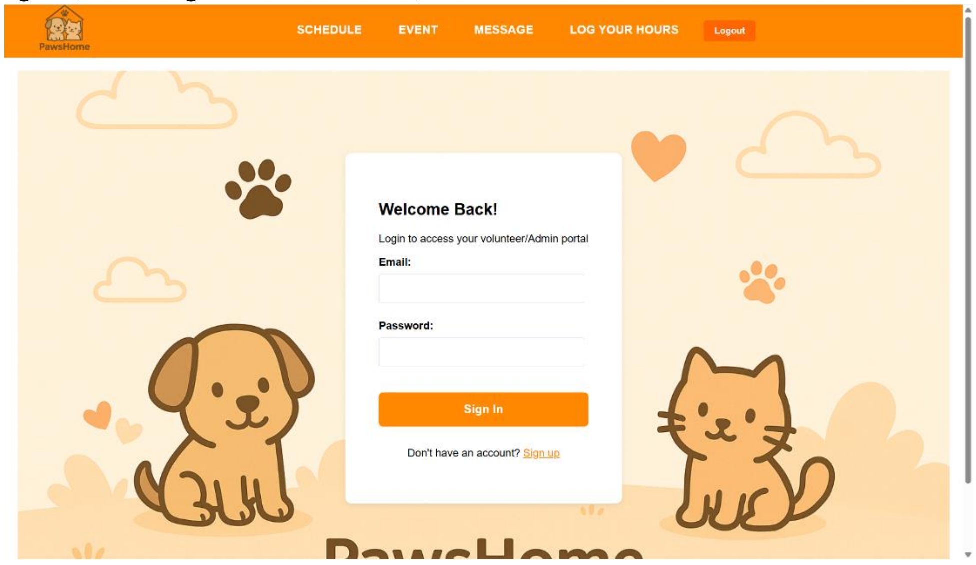
Phone: Enter your phone number

Password: Enter your password

Confirm Password: Confirm your password

Create Account

(3) Login: Sign in; admins go to the dashboard, volunteers to schedule.



Welcome Back!

Login to access your volunteer/Admin portal

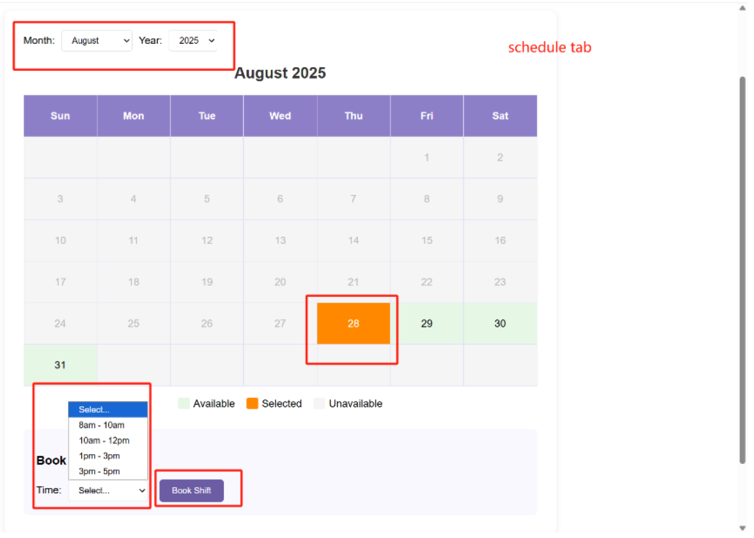
Email:

Password:

Sign In

Don't have an account? Sign up

(4) Schedule: Book a volunteer shift on a selected date and time.



Month: August Year: 2025

schedule tab

August 2025

Sun	Mon	Tue	Wed	Thu	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

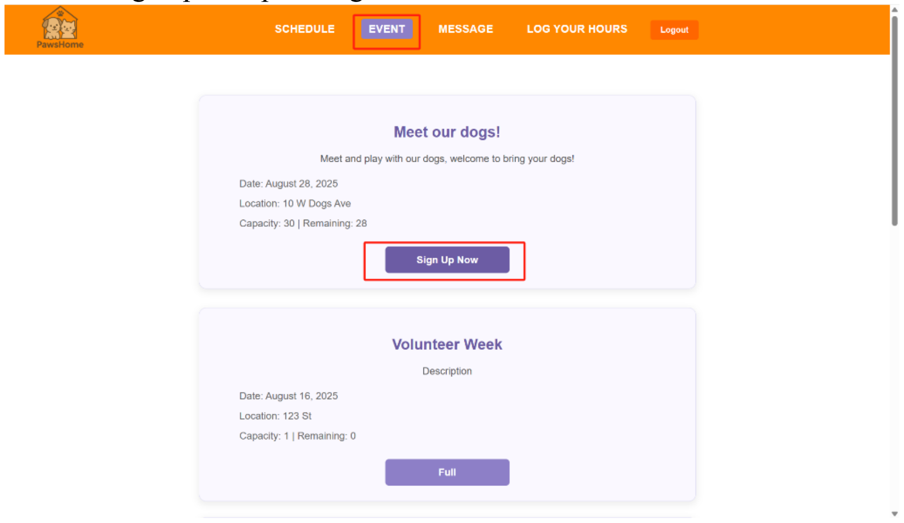
🟩 Available 🟧 Selected ⬜ Unavailable

Book Time: Select...

Select...
8am - 10am
10am - 12pm
1pm - 3pm
3pm - 5pm

Book Shift

(5) Events: Browse and sign up for upcoming events.



Meet our dogs!

Meet and play with our dogs, welcome to bring your dogs!

Date: August 28, 2025

Location: 10 W Dogs Ave

Capacity: 30 | Remaining: 28

Sign Up Now

Volunteer Week

Description

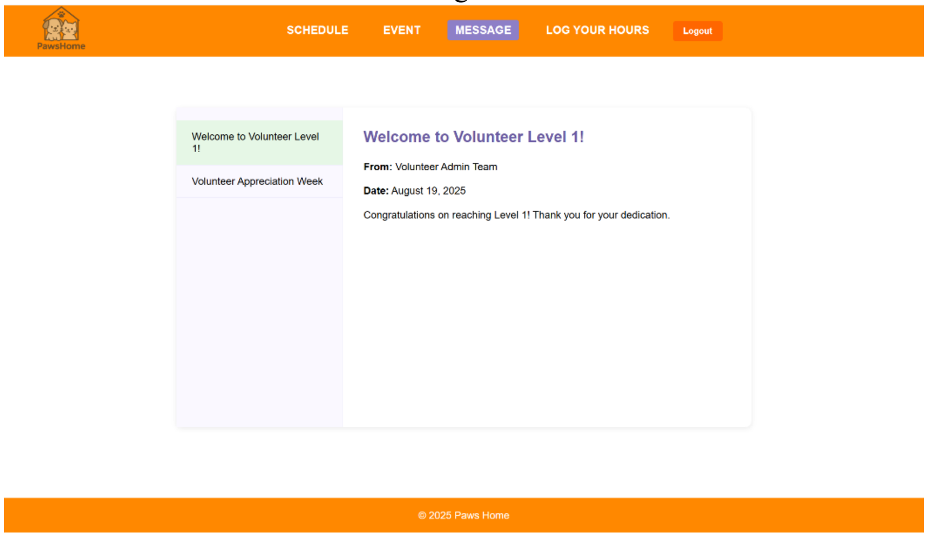
Date: August 16, 2025

Location: 123 St

Capacity: 1 | Remaining: 0

Full

(6) Message: View announcements and direct messages.



Welcome to Volunteer Level 1!

Volunteer Appreciation Week

Welcome to Volunteer Level 1!

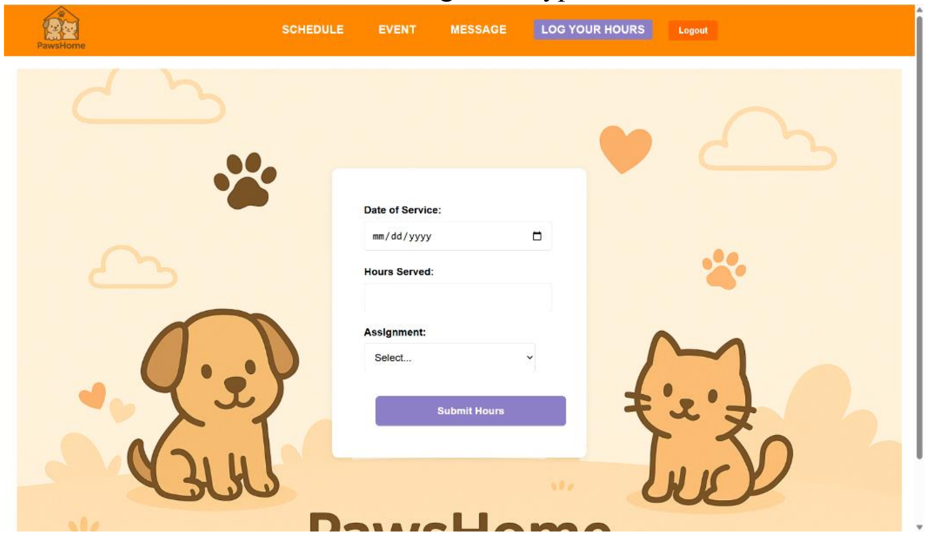
From: Volunteer Admin Team

Date: August 19, 2025

Congratulations on reaching Level 1! Thank you for your dedication.

© 2025 Paws Home

(7) Log Your Hours: Submit hours served with assignment type.



Date of Service:

mm/dd/yyyy

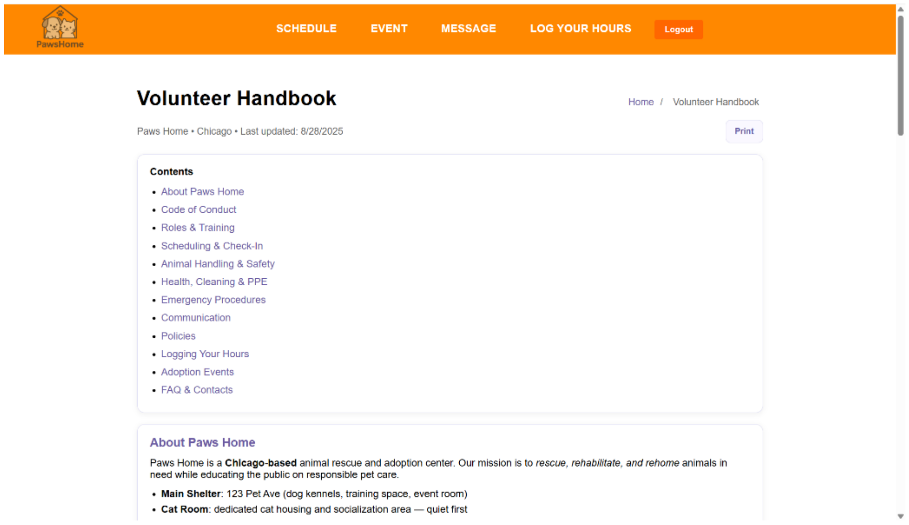
Hours Served:

Assignment:

Select...

Submit Hours

(8) Handbook: Policies and volunteer guide.



Volunteer Handbook

Paws Home • Chicago • Last updated: 8/28/2025

Contents

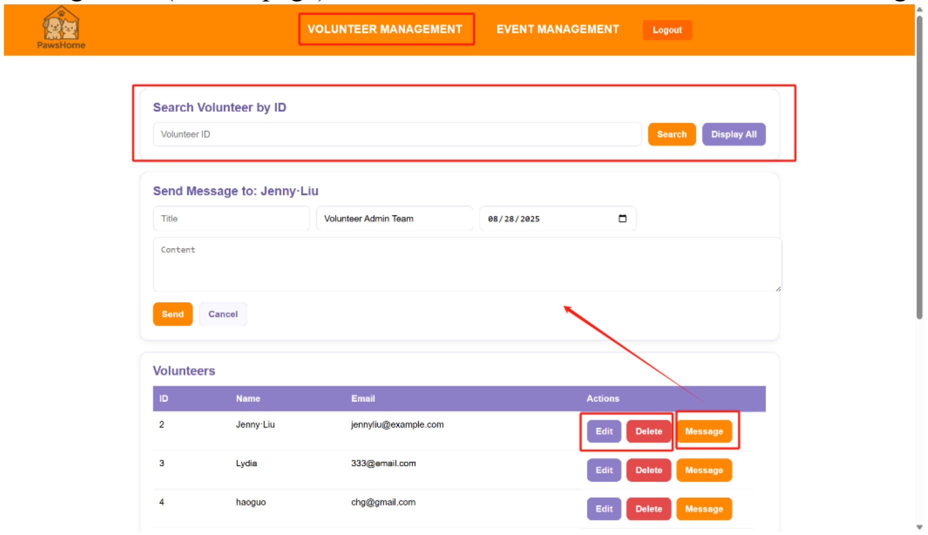
- About Paws Home
- Code of Conduct
- Roles & Training
- Scheduling & Check-In
- Animal Handling & Safety
- Health, Cleaning & PPE
- Emergency Procedures
- Communication
- Policies
- Logging Your Hours
- Adoption Events
- FAQ & Contacts

About Paws Home

Paws Home is a **Chicago-based** animal rescue and adoption center. Our mission is to *rescue, rehabilitate, and rehome* animals in need while educating the public on responsible pet care.

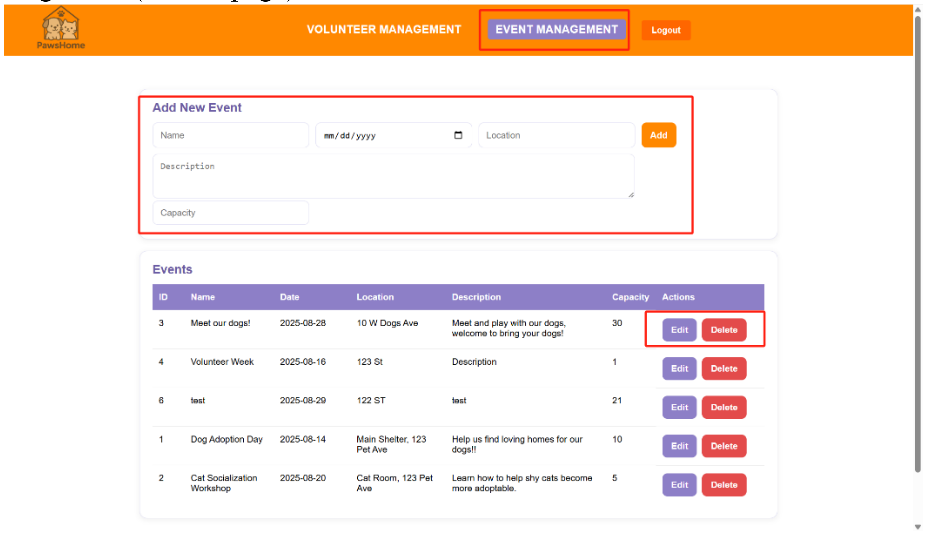
- **Main Shelter**: 123 Pet Ave (dog kennels, training space, event room)
- **Cat Room**: dedicated cat housing and socialization area — quiet first

(9) Volunteer Management (Admin page): Search, edit, delete volunteers, and send messages.



ID	Name	Email	Actions
2	Jenny-Liu	jennyliu@example.com	Edit Delete Message
3	Lydia	333@email.com	Edit Delete Message
4	haoguo	chg@gmail.com	Edit Delete Message

(10) Event Management (Admin page): Create, edit, or remove events.



ID	Name	Date	Location	Description	Capacity	Actions
3	Meet our dogs!	2025-08-28	10 W Dogs Ave	Meet and play with our dogs, welcome to bring your dogs!	30	Edit Delete
4	Volunteer Week	2025-08-16	123 St	Description	1	Edit Delete
6	test	2025-08-29	122 ST	test	21	Edit Delete
1	Dog Adoption Day	2025-08-14	Main Shelter, 123 Pet Ave	Help us find loving homes for our dogs!!	10	Edit Delete
2	Cat Socialization Workshop	2025-08-20	Cat Room, 123 Pet Ave	Learn how to help shy cats become more adoptable.	5	Edit Delete

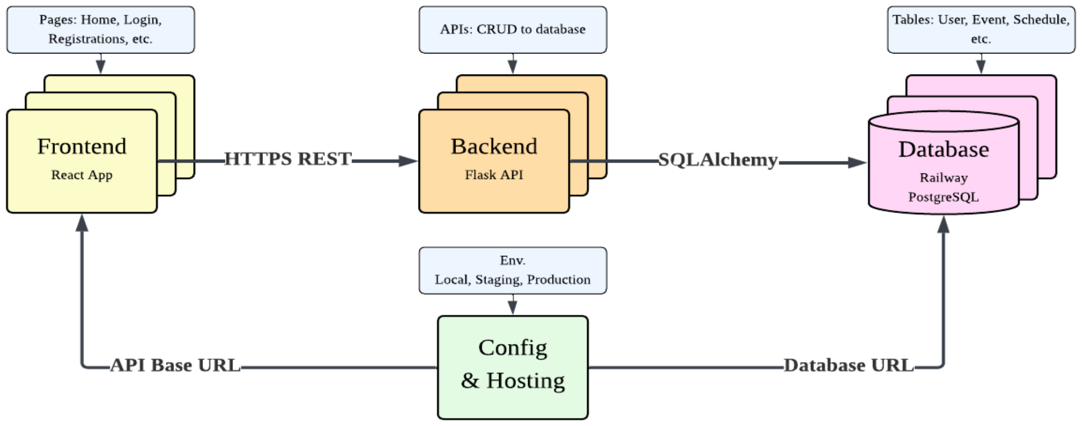
4.3 Main workflows

(1) Create account: Fill Register form → Submit → Redirected to Schedule.
(2) Sign in: Enter email & password → Redirects based on role.
(3) Book shift: Pick date in Schedule → Choose time slot → Confirm.
(4) Sign up for events: Select event → Click "Sign Up Now."
(5) Read messages: Open Message → Click to view details.
(6) Log hours: Enter date, hours, type → Submit
(7) Admin tasks: Manage volunteers and events with forms.

4.4 Known limitations

- Passwords are stored in plain text (test use only).
- No UI to promote admins; must update the database directly.
- Schedule doesn't show existing bookings or prevent overlaps.

5 Architecture Diagram



The system architecture consists of four main components:

(1) Frontend (React App)

- Provides the user interface for volunteers and administrators.
- Includes pages for Home, Login, Register, Schedule, Events, Messages, Log Hours, Handbook, and Admin dashboards.
- Communicates with the backend via HTTPS REST APIs.

(2) Backend (Flask API)

- Acts as the middle layer between frontend and database.
- Exposes REST endpoints for login, registration, volunteer management, schedules, events, messages, and hours logging.
- Uses SQLAlchemy ORM to interact with the PostgreSQL database.
- CORS is enabled to allow the frontend to make requests.

(3) Database (PostgreSQL, hosted on Railway or local)

- Stores persistent data, including users, volunteers, admins, schedules, events, signups, messages, recipients, and logged hours.
- Accessed only via the backend API to ensure security and consistency.

(4) Config & Hosting

- Application behavior is controlled through environment variables:
- REACT_APP_API_URL connects the frontend to the backend.
- DATABASE_URL connects the backend to the database.
- Supports multiple environments: Local (development), Staging (testing), and Production (live).

6 Additional Section

6.1 Deployment Pipeline Overview (CI/CD)

(1) Code Commit (GitHub Repository)

- Developers push frontend (React) and backend (Flask) code to GitHub.
- Each push triggers an automated workflow.

(2) Build & Test

- Frontend: Dependencies installed, React build generated.
- Backend: Flask dependencies installed, unit tests executed.
- Database migrations: Run with Flask-Migrate (if applicable).

(3) Deployment

- Successful builds are auto-deployed to Railway.
- Railway provisions:

  React frontend → Hosted as static build.

  Flask backend → Containerized service.

  PostgreSQL database → Managed service.

(4) Monitoring & Rollback

- Railway logs allow quick debugging.
- If deployment fails, rollback can be done by:

  Redeploying previous successful build.

  Switching environment variable back to last stable version.

6.2 Security Considerations

6.2.1 Authentication & Authorization

Authentication:

- Users log in with email + password.
- Session-based or token-based authentication (JWT) supported.

Authorization:

- Role-based access control (RBAC):

  Admin: Manage events, shifts, and volunteers.

  Volunteer: Register for events, view personal schedule.

- Enforced at API layer (Flask routes protected by decorators).

6.2.2 Data Security

Sensitive information (DB password, API keys) stored in Railway environment variables, not hardcoded.