Implementation and Testing for

Neighborly

Prepared By: Ariel Brigantty
UPRRP
v1.0
12/12/2025

Table of Contents

Version History

Date	Author	Version	Description
11/30/2025	Ariel Brigantty	0.1	First draft.
12/08/2025	Ariel Brigantty	0.2	Second draft.
12/09/2025	Ariel Brigantty	0.9	Format polishing, double-checking.
12/10/2025	Ariel Brigantty	1.0	Final draft.

1 Introduction

1.1 Purpose

The Neighborly Integration Manual aims to explain the integration of Neighborly at a high-level and highlight important implementation details that may be relevant to developers working on it, integrating it into a new system or integrating it into a preexisting system. It covers technical specifications, deployment instructions, access credentials for test accounts, and testing procedures to ensure successful system integration and operation.

Document Conventions

Bold: Used for section headings and important terminology

`Monospace`: Used to differentiate code from text

1.2 Intended Audience and Reading Suggestions

- **System Administrators**: should read Section 2 (Implementation Details) for deployment instructions, which may also be relevant during maintenance.
- **Quality Assurance Teams**: should read Section 3 (Test Cases) for instructions on expected values and behaviors for validation.
- **Developers**: should read the entire document for a comprehensive understanding of the Neighborly platform.
- **Project Managers**: should read Section 2.3 for specifications regarding the various test accounts, as these may prove relevant when assigning privileges to their team's accounts.

2 Implementation Details

The Implementation Details section provides technical specifications and deployment instructions for the Neighborly platform. It outlines the complete hosting environment, including web server configuration on a local machine, how to setup the MySQL database Neighborly hinges on to run, and the integration between PHP backend operations and the database. The documentation covers the step-by-step process for deploying both the application files and database schema, along with maintenance procedures to ensure ongoing system maintenance is easy. All user-facing features are delivered through a web-based interface that connects to a MySQL backend following MVC architecture principles.

2.1 Technical Specifications

On application hosting:

- Primary hosting on **l**ocal machine
- Web server: PHP webserver
    - Its demo is fully hosted on the ADA platform through the following website: https://ada.uprrp.edu/~raffaella.comes/Neighborly/pages/login.php

On the database system:

- MySQL 8.* database server
- Hostable locally or through a server
- All user data, events, skills, and relationships stored in MySQL tables
- Database name: neighborly

On database integration:

- All user interactions, at some level, trigger MySQL database operations
- User registration → `INSERT` into `USERS` table
- Login authentication → `SELECT` queries against users table with PHP hashing
- Event management → `UPDATE`, `INSERT` and `DELETE` operations on events and related tables
- Profile updates → `UPDATE` queries on `USERS` and profile tables
- Opportunity browsing → `SELECT` queries against `SKILLS` table and comparisons against record on `EVENTS'` table skills

On programming languages and development methods:

- **Frontend**: HTML5, CSS3, Vanilla JavaScript
- **Backend**: PHP 8.*
- **Architecture**: Model-View-Controller (MVC) pattern
- **Authentication**: Single session-based (cookieless, as user information is stored in the database) with SHA-256 password hashing

2.2 How to Run the Application

On configuring the local environment:

1. **Prerequisites Installation:**
    - Install PHP 8.* or higher on your local machine
    - Install MySQL 8.* Community Server
    - Ensure PHP has the mysqli extension and MySQL driver enabled
        - In the event Neighborly is being hosted on Windows, access the PHP installation php.ini file, found in PHP's native installation folder, and enable the driver by removing the semicolon on the line: `;extension=pdo_mysql.`
2. **PHP Environment Verification:**
    - Open terminal/command prompt and verify PHP installation: `php -v`
    - Check for mysqli extension: `php -m | grep mysqli` (Linux/Mac) or `php -m | findstr mysqli` (Windows)
        - If mysqli is not enabled, edit your php.ini file and uncomment the line: `;extension=mysqli`
3. **Local Server Setup:**
    - Navigate to the Neighborly project directory in terminal/command prompt
    - Start PHP's built-in web server: `php -S localhost:8000`
    - The application will be accessible at `http://localhost:8000`

- To use a different port, change the port number: `php -S localhost:3000`

On implementing the database:

1. **MySQL Database Creation**:
   - Run a MySQL Community server through the terminal using `net start MySQL80`
   - Create a new MySQL database named neighborly
2. **Schema Implementation**:
   - Execute the complete database schema SQL script to create all tables, found in the neighborly repository hosted in Github under the database folder.
     - This setup assumes that the default user is `root`. If it is not, implement the correct credentials in the database.php script under the pages folder.
   - Run the dummy data insertion script to populate initial test data, or exclusively import the schema.
   - Verify all tables, relationships, and indexes are properly created using the `DESCRIBE` keyword on MySQL.
3. **Database Connection**:
   - Configure PHP database connection strings with correct host, username, password, and database name.
     - These come preconfigured for localhost on port 3306, the default for MySQL Community servers.
       - **NOTE**: If Neighborly is hosted off a Linux or iOS system, configure MySQL to remove case-sensitivity by accessing your MySQL installation's my.cnf file and setting `lower_case_table_names` to 1. The MySQL database was written in a Windows environment, where queries are not case sensitive by default. This differs from a Linux or iOS environment, where the value is set to 0 by default.
       - Corroborate through `SELECT 'A' = 'a';` which must return an exit code of 1.
   - Test database connectivity from the application using `SELECT` queries through a MySQL Workbench or the terminal.
4. **Application Deployment**:
   - **File Upload**: Transfer all PHP, HTML, CSS, and JavaScript files to the web server's public directory, or the directory the port its being hosted on references.
   - **Configuration**: Update application configuration files with correct database credentials and server paths, which may be unique to the server.
   - **Permissions**: Set appropriate file permissions for uploads and temporary directories. The database should have global permissions, as the front-end will itself limit the queries possible by the user.
5. **Server Access**:
   - Once deployed, users access the application via its configured domain URL.
   - The login screen serves as the entry point prior to being able to access user features, regardless of role.
   - All features function through the web interface of a modern web browser.
6. **Maintenance Suggestions**:

- Regular database backups done manually, through automated scripts or the host panel of the cloud hosting service being used.
- Monitor server resources and performance metrics through PHP's analysis tools or through the cloud hosting service's host panel.
- Apply security updates to PHP, MySQL, and server operating system, if ever required.

2.3 Credentials

Test Credentials

- Volunteer: username maria_garcia / password password123
- Organizer: username comunidad_pr / password orgpass123
- Administrator: username admin_neighborly / password adminpass123
  - **NOTE**: new credentials for volunteers and organizers may be made through the registration screen on a functioning Neighborly deployment. If one is not available, they may be directly inserted into the MySQL database through the `USER` and `VOLUNTEER`, `ORGANIZER` and `ADMIN` tables. The latter is the only method for creating administrator accounts.

3 Test Cases

The following test cases are can be validated through MySQL queries and interaction with a live web server.

3.1 Test Case – User Login and Registration

1. Navigate to Neighborly and access login screen	Login form displays with options for existing users and new registration.
2. Click "Click here!" and complete registration form	MySQL database adds new USERS record.
3. Login with newly created credentials	MySQL verifies hashed credentials, grants access to role-specific timeline.
4. Login with invalid credentials	MySQL query returns no match and web server denies login.
5. Login with existing test account	Successful authentication, redirect to role-specific timeline.

3.2 Test Case – Volunteer Timeline Population

1. Login as volunteer	MySQL populates timeline with events approved by administration.
2. View opportunity timeline	Chronological event list (based on event submission) displays, with matches being highlighted.
3. Click "Volunteer" for an event	MySQL inserts record into `EVENT_ATTENDEES` table, slot count updates.
4. Check "Profile" section	MySQL query returns volunteer profile with relevant information from `EVENT_ATTENDEES` and `VOLUNTEERS`, `USERS` and `USER_SKILLS`.
5. Update volunteer profile	MySQL updates relevant record under `USERS`, `EVENT_ATTENDEES` and `USER_SKILLS`, timeline suggestions refresh accordingly.

3.3 Test Case – Organizer Event Management

1. Login as organizer and create new event	MySQL inserts event record with organizer ID, status pending.
2. Edit existing event details	MySQL updates `EVENTS` table of a `event_id` matching that found on the event listing.
3. View event registrations	MySQL query returns volunteer list from `EVENT_ATTENDEES` table that matches the with the event's `event_id`.
4. Answer volunteer questions	MySQL updates `QUESTIONS` table with `answers_text` and `organizer_ID`.
5. Delete an event	MySQL removes event from the `EVENTS` table.

3.4 Test Case – Admin Event Approval

1. Login as admin and view pending events	MySQL query returns events with `is_approved = FALSE`
2. Approve a pending event	MySQL updates event with `is_approved = TRUE`
3. Reject a pending event	MySQL may delete event or mark as rejected based on implementation.
4. View approved events	MySQL query returns events with `is_approved = TRUE`
5. Restore rejected event	MySQL query returns `is_approved = TRUE` after `FALSE`, assuming the event was not deleted.

3.5 Test Case – Profile Management

1. Login and navigate to profile page	MySQL returns relevant record under `USERS`, `EVENT_ATTENDEES` and `USER SKILLS`.
2. Update personal information	MySQL updates `USERS`, `VOLUNTEERS` and `ORGANIZERS` tables, where applicable.
3. Add or remove skills	MySQL modifies `USER_SKILLS` table relationships.
4. Change location preference	MySQL updates `USERS` record.
5. View profile changes	MySQL query confirms updates visible on profile.

4 Bugs and Fixes

4.1 Duplicate Event Sign-ups

Bug: The system allowed the same volunteer to register multiple times for the same event, creating duplicate registrations in the database.
Technical Details:

- **Root Cause**: Missing unique constraint on the `EVENT_ATTENDEES` table for the combination of `EVENT_ID` and `VOLUNTEER_ID`
- **Database Impact**: Multiple identical records in EVENT_ATTENDEES table for same user-event combination
- **User Impact**: Volunteers could accidentally register multiple times, skewing attendance counts and event capacity calculations

Solution:

- Added composite UNIQUE constraint on (`EVENT_ID, VOLUNTEER_ID`) columns in `EVENT_ATTENDEES` table
- Added application-level check before attempting registration

4.2 Password Security

Bug: User passwords were stored in plain text in the database, creating severe security vulnerabilities.
Technical Details:

- **Root Cause**: Passwords stored as plain text in `USERS.password` column
- **Security Risk**: Database breach would expose all user credentials directly
- **Compliance Issue**: Violated basic security best practices and potential privacy regulations

Solution:

- Implemented SHA-256 cryptographic hashing for all passwords
- Renamed `USERS.password` to `USERS.password_hash`
- Updated authentication system to compare hashed values instead of plain text
- Existing passwords were reset on the test account

4.3 Session Timeout

Bug: User sessions expired after 30 minutes of inactivity, causing frustration during extended browsing or event registration.
Technical Details:

- **Root Cause**: PHP session configuration with `session.gc_maxlifetime = 1800` (30 minutes)
- **Server Impact**: Premature session garbage collection
- **User Impact**: Users logged out unexpectedly, losing unsaved form data

Solution:

- Configured `session.gc_maxlifetime = 7200` (2 hours) in php.ini
  - Developers must make this edit manually
- Added session persistence for critical actions like form submissions

4.4 MySQL User Privileges

Bug: The database user account lacked necessary permissions for write operations, preventing organizers from creating and managing events.
Technical Details:

- **Root Cause**: Database user granted only `SELECT` privileges during initial setup
- **Error Messages**: "`INSERT` command denied to user", "`UPDATE` command denied", "`DELETE` command denied"
- **Impact**: All roles could view but not create, edit, or delete events; system read-only for content creation

Solution:

- Granted comprehensive privileges: `SELECT`, `INSERT`, `UPDATE`, `DELETE` on all Neighborly tables
- Verified permissions work for all CRUD operations across user roles

4.5 Server File Upload Restrictions

Bug: System restrictions prevented developers from uploading to the Namecheap server.
Technical Details:

- **Root Cause**: Multiple layered restrictions:
  - PHP: upload_max_filesize = 2M and post_max_size = 8M too small
  - File permissions: Upload directory had 644 permissions (read-only for web server)
  - Restrictions on ADA server
- **Error Messages**: "Cannot save uploaded file", "File upload size exceeded", "File type not allowed"

Solution:

- **PHP Configuration**: Replaced all file-keeping, namely with the images, to links, effectively outsourcing storage.

4.6 Login Errors

Bug: Attempts to log in on Linux machines were failing.
Technical Details:

- **Root Cause**: The Windows default configuration of the MySQL differs from the Linux default configuration in that case-sensitivity is default on Linux, but not on Windows.
- **Error Messages**: These were very general, as we implemented a try-catch that was generalized for all exceptions.

Solution:

- **MySQL Configuration**: Included details in the Implementation Manual in regards to the case-sensitivity and how it differs across operating systems, empowering system developers to change the variable contained in my.cnf to meet their needs.

5 References

- CSS Official Documentation
- JavaScript Official Documentation
- Mozilla MDN Web Docs
    - AJAX (Asynchoronous Javascript and XML) Glossary
- MySQLi Imrpvoed Extension Official Documentation
- PHP Official Documentation
    - MySQLi Imrpvoed Extension Official Documentation
- MySQL 8.0 Reference Manual