NeighbourNet.

Submitted to the

Department of Master of Computer Applications

In partial fulfillment of the requirements

For Full Stack Development(MCAE31)

By

Nithin **(1MS23MC063)**

Pramod Upadhyaya **(1MS23MC071)**

Saathvik N Sharma **(1MS23MC084)**

Under the Guidance of

Sushitha Bhat

Assistant Professor

Department of Master of Computer Applications

RAMAIAH INSTITUTE OF TECHNOLOGY

(Autonomous Institute, Affiliated to VTU)

Accredited by National Board of Accreditation & NAAC with 'A+' Grade

MSR Nagar, MSRIT Post, Bangalore-560054

www.msrit.edu

2024

DEPARTMENT OF
MASTER OF COMPUTER APPLICATIONS

CERTIFICATE

This is to certify that the project entitled NeighbourNet. carried out by

1) Nithin bearing USN 1MS23MC063
2) Pramod Upadhyaya USN 1MS23MC072
3) Saathvik N Sharma USN 1MS23MC084

students of II semester, in partial fulfillment for Full Stack Development (MCAE31) during the academic year 2024-2025.

Guide	**Head of Department**

Name of the Examiner	**Sign with Date**
1	
2	

ABSTRACT

NeighborNet is a community-focused web application designed to facilitate neighborhood interaction and support through essential features. Built with React.js for the frontend, Node.js and Express for the backend, and MongoDB for data management, the application aims to provide a user-friendly platform for basic community engagement. The core functionalities include user profile management, help requests, and resource sharing.

The application allows users to create and manage their profiles, submit requests for help or support within their community, and share or access local resources. By focusing on these fundamental features, NeighborNet aims to simplify neighborhood coordination and enhance community connections. The design emphasizes ease of use, ensuring that even users with minimal technical skills can navigate the platform effectively. The implementation also prioritizes responsiveness and performance, making it accessible on various devices and ensuring a seamless user experience.

The main challenge addressed by NeighborNet is providing a straightforward and efficient tool for local engagement, avoiding the complexity found in more feature-rich platforms. The result is a streamlined application that supports essential community interactions, with a focus on reliability and simplicity.

TABLE OF CONTENTS

1. INTRODUCTION

NeighborNet is a web-based platform designed to enhance neighborhood connectivity by fostering interactions and support among community members. Built with a focus on simplicity and accessibility, the platform serves as a digital hub for local engagement, providing a space where neighbors can communicate, collaborate, and assist one another with ease. Its design caters to users of all technical backgrounds, ensuring that even those with limited digital experience can participate in community activities without barriers.

The platform's primary goal is to promote a sense of belonging and mutual aid within neighborhoods. In today's fast-paced, digitally driven world, NeighborNet seeks to bridge the gap between physical proximity and digital connectivity, making it easier for neighbors to connect and collaborate. By providing an intuitive, easy-to-use interface, it encourages residents to take a more active role in their communities, whether by offering help, sharing resources, or simply getting to know the people around them better.

NeighborNet was developed using modern web technologies, including React.js for the frontend, Node.js, and Express for the backend, with MongoDB managing data storage. These technologies provide a responsive, scalable, and secure foundation for the platform, enabling seamless interactions and ensuring that the platform remains reliable and performant across a range of devices.

The platform emphasizes privacy and security, with features like email-based verification ensuring that user data is protected. By focusing on core community-building features, NeighborNet avoids the complexity of larger platforms, making it an ideal solution for those seeking a simple, effective way to engage with their neighbors.

2. LITERATURE REVIEW

2.1 Existing Systems

Numerous platforms aim to foster local community engagement, each with its unique strengths and weaknesses. Examples include Nextdoor, Facebook Groups, and specialized apps like the Buy Nothing Project. These systems vary in scope from comprehensive neighborhood networks to focused resource-sharing communities, highlighting diverse approaches to enhancing local interaction. Few of those platforms include:

i. Nextdoor

Strengths:

- **Wide Reach:** Nextdoor connects a broad range of users within specific neighborhoods, fostering a strong sense of local community.
- **Comprehensive Features:** Offers various functionalities such as local news, safety updates, recommendations, and classifieds.
- **High Engagement:** Promotes active participation through local discussions and events.

Lacks:

- **Cluttered Experience:** The breadth of features can lead to a crowded and sometimes overwhelming user interface.
- **Privacy Concerns:** Users have raised concerns about data privacy and security.
- **Limited Focus on Resource Sharing:** While it includes resource sharing, it's not as centralized or streamlined as dedicated platforms.

ii. Facebook Groups (Local)

Strengths:

- **Versatility:** Facebook Groups can be tailored for any local interest or need, with high user engagement due to Facebook’s large user base.
- **Integrated Features:** Combines social networking with group interactions, such as events and discussions.

Lacks:

- **Complexity:** The platform's broad features and ads can detract from the localized focus.
- **Privacy Issues:** Concerns over data privacy and Facebook’s use of personal data.

- **Fragmented Interaction:** Local groups may get lost among numerous other types of content on Facebook.

iii. Buy Nothing Project

Strengths:

- **Focused on Resource Sharing:** Emphasizes the exchange of free goods within communities, promoting sustainability and neighborly support.
- **Strong Community Aspect:** Encourages close-knit community interactions with a focus on sharing and giving.

Lacks:

- **Limited Scope:** Focuses solely on resource sharing, lacking broader functionalities such as help requests or community event management.
- **Variable Engagement:** Success and activity levels can vary greatly between different local groups.

iv. MyCoop

Strengths:

- **Targeted Communication:** Facilitates communication within specific buildings or complexes, making it ideal for property management and internal community issues.
- **Focused Features:** Tailored to address needs specific to shared living environments.

Lacks:

- **Narrow Scope:** Limited to apartment or building management, which doesn't extend to broader neighborhood or community support.
- **Less Versatility:** Fewer features for diverse neighborhood interactions and external community engagement.

v. Meetup (Local)

Strengths:

- **Event-Centric:** Excels in organizing and finding local events based on interests, fostering real-world connections.
- **User-Friendly:** Easy to use for discovering and joining local activities.

Lacks:

- **Limited Community Support:** Focuses mainly on events rather than comprehensive neighborhood interaction or support.
- **Event-Driven:** Does not cater to other aspects like help requests or resource sharing.

vi. OLIO

Strengths:

- **Resource Efficiency:** Focuses on reducing waste by sharing surplus food and goods, promoting sustainability and local resource utilization.
- **Local Focus:** Encourages close community interactions around food and household items.

Lacks:

- **Restricted Functionality:** Primarily centered on food and item sharing, without features for broader neighborhood support or help requests.
- **Limited User Engagement:** May not offer enough variety in interactions and functionalities to fully engage a diverse community.

2.2 Proposed System

NeighborNet distinguishes itself from existing community platforms by focusing on simplicity, user-friendliness, and essential functionality. Unlike broader social media sites or complex neighborhood apps, NeighborNet offers a streamlined interface specifically designed for efficient local engagement. The platform prioritizes core features such as user profile management, help requests, and resource sharing, ensuring that interactions remain straightforward and relevant to the community's needs.

Key innovations include a straightforward OTP verification process using Nodemailer and Mailtrap, which enhances security and user verification without adding unnecessary complexity. Additionally, the platform is designed to be highly accessible across devices, emphasizing performance and ease of use. By removing the complexity of features found in more extensive systems and focusing on the fundamental aspects of community engagement, NeighborNet offers a more focused, reliable, and intuitive tool for local interaction. This approach aims to foster stronger neighborhood connections and simplify coordination, setting it apart from the more generalized or feature-heavy alternatives.

3. TECHNOLOGIES USED

NeighborNet employs a suite of modern technologies to deliver a robust and efficient community platform. Here’s a detailed overview of the key technologies utilized in the development of NeighborNet, along with their key features:

1. **React.js:**
    - **Frontend Framework:** React.js is used for building the dynamic and responsive user interface of NeighborNet.
    - **Component-Based Architecture:** Facilitates the creation of reusable UI components, enhancing development efficiency and maintainability.
    - **Virtual DOM:** Improves performance by minimizing direct manipulation of the DOM, ensuring faster updates and rendering.
    - **Declarative UI:** Allows developers to design user interfaces by specifying what the UI should look like at any given state, making the code more predictable and easier to debug.
2. **Node.js:**
    - **Runtime Environment:** Node.js enables server-side development with an event-driven, non-blocking I/O model, ideal for handling multiple simultaneous connections.
    - **Scalability:** Supports scalable network applications, making it suitable for applications that require real-time updates and high performance.
    - **Single Programming Language:** Allows both client-side and server-side code to be written in JavaScript, streamlining development and maintenance.
3. **Express:**
    - **Web Application Framework:** Express simplifies server-side development with its minimalistic approach, providing essential features for building web applications and APIs.
    - **Middleware Support:** Facilitates the use of middleware for handling requests, responses, and errors, improving code organization and modularity.
    - **Routing:** Provides a flexible routing system to define and manage different API endpoints and HTTP methods efficiently.

4. **MongoDB:**
    - **NoSQL Database:** MongoDB uses a document-oriented model, offering flexibility in data storage and structure.
    - **Scalability:** Handles large volumes of data with ease, making it suitable for applications with varying data requirements.
    - **Rich Query Language:** Supports complex queries and indexing, allowing for efficient data retrieval and manipulation.
5. **Nodemailer and Mailtrap:**
    - **Nodemailer:** A module for sending emails from Node.js applications, supporting various email services and protocols.
    - **Mailtrap:** Provides a secure and reliable environment for testing email functionalities during development, preventing accidental email dispatches to real users.
    - **SMTP Integration:** Nodemailer allows integration with SMTP servers, while Mailtrap offers email testing without affecting production systems.
6. **Helmet:**
    - **Security Middleware:** Helmet enhances security by setting various HTTP headers to protect against common web vulnerabilities, such as cross-site scripting (XSS) and clickjacking.
    - **Customizable:** Offers a range of security policies that can be tailored to meet the specific needs of the application.
7. **xss-clean:**
    - **Input Sanitization:** Protects against cross-site scripting (XSS) attacks by filtering and sanitizing user inputs before processing.
    - **Middleware Integration:** Easily integrates with Express applications to ensure that all incoming data is cleaned and safe.

8. **express-rate-limit:**
    - **Rate Limiting:** Limits the number of requests a user can make to the server within a specified time frame, preventing abuse and ensuring fair usage.
    - **Customizable Limits:** Allows configuration of request limits based on various parameters, such as IP address and time window.
9. **Material-UI:**
    - **UI Component Library:** Material-UI provides a set of pre-designed components that follow Google's Material Design guidelines, enhancing the visual appeal and usability of the application.
    - **Customizable Themes:** Allows developers to customize themes and styles to match the branding and design requirements of the project.
    - **Responsive Design:** Ensures that the user interface is adaptable to different screen sizes and devices, improving accessibility and user experience.
    - **Component Richness:** Offers a wide range of components, such as buttons, dialogs, and forms, that streamline development and provide a consistent look and feel across the application.

4. IMPLEMENTATION

The implementation of NeighborNet integrates various technologies and tools to deliver a robust and user-friendly platform for community engagement. Below is a sample of code snippets that illustrate the core components and functionalities of the application.

4.1 Sample Code

i. Server Setup (Node.js and Express)

The server setup involves initializing an Express application and configuring middleware for handling requests and connecting to MongoDB

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();
const helmet = require('helmet');
```

```
const rateLimit = require('express-rate-limit');
const xss = require('xss-clean');

const app = express();
const port = process.env.PORT || 8000;

if (!process.env.MONGO_URI) {
    console.error('Missing MONGO_URI environment variable');
    process.exit(1);
}

app.use(cors());
app.use(express.json());
app.use(helmet());
app.use(xss());

const limiter = rateLimit({
    windowMs: 15 * 60 * 1000,
    max: 100
});
app.use(limiter);

mongoose.connect(process.env.MONGO_URI).then(
    () => {
        console.log("MongoDB Connected!");
    }
).catch((err) => {
    console.error('MongoDB Connection error: ' + err);
});

app.use('/api/users', require('./routes/users'));
app.use('/api/help-requests',
require('./routes/helpRequests'));
app.use('/api/resources', require('./routes/resources'));
app.use('/api/events', require('./routes/events'));
app.use('/api/volunteers', require('./routes/volunteers'));
app.use('/api/reviews', require('./routes/reviews'));
app.use('/api/notifications',
require('./routes/notifications'));
app.use('/api/messages', require('./routes/messages'));
app.use('/api/settings', require('./routes/settings'));
app.use('/api/admin', require('./routes/admin'));
app.use('/api/analytics', require('./routes/analytics'));

app.use((err, req, res, next) => {
```

```
    console.error(err.stack);
    res.status(500).json({ message: 'Something went wrong!' });
});

const server = app.listen(port, () => {
    console.log(`Server running on port ${port}`);
});

const shutdown = (signal) => {
    console.log(`Received ${signal}. Closing HTTP server
gracefully...`);

    server.close(() => {
        console.log('HTTP server closed.');
        mongoose.connection.close(() => {
            console.log('MongoDB connection closed.');
            process.exit(0);
        });
    });

    setTimeout(() => {
        console.error('Forcing server shutdown.');
        process.exit(1);
    }, 10000);
};

process.on('SIGTERM', () => shutdown('SIGTERM'));
process.on('SIGINT', () => shutdown('SIGINT'));
```

ii. User Route for Registration (Express Router)

This route handles user registration and OTP verification.

```
const express = require('express');
const router = express.Router();
const UserController = require('../controllers/userController');

router.post('/register', UserController.register);
router.post('/verify-otp', UserController.verifyOTP);
router.post('/resend-otp', UserController.resendOTP);
router.post('/forgot-password', UserController.forgotPassword);
router.post('/reset-password', UserController.resetPassword);

module.exports = router;
```

iii. Sample User Controller Method (Node.js)

The controller handles the logic for user registration.

```
const User = require('../models/User');
const OTPService = require('../services/otpService');

exports.register = async (req, res) => {
    try {
        const { email, password } = req.body;
        if (!email || !password) {
            return res.status(400).json({ message: 'Email and
password are required.' });
        }

        // Check if user already exists
        const existingUser = await User.findOne({ email });
        if (existingUser) {
            return res.status(400).json({ message: 'User already
exists.' });
        }

        // Create new user
        const newUser = new User({ email, password });
        await newUser.save();

        // Send OTP
        const otp = OTPService.generateOTP();
        OTPService.sendOTP(email, otp);

        res.status(201).json({ message: 'User registered
successfully. OTP sent.' });
    } catch (err) {
        res.status(500).json({ message: 'Server error.' });
    }
};
```

iv. Frontend Component (React)

A React component to display user profiles.

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const UserProfile = ({ userId }) => {
    const [user, setUser] = useState(null);

    useEffect(() => {
```

```
        const fetchUser = async () => {
            try {
                const response = await
axios.get(`/api/users/${userId}`);
                setUser(response.data);
            } catch (error) {
                console.error('Error fetching user profile:',
error);
            }
        };

        fetchUser();
    }, [userId]);

    if (!user) return <div>Loading...</div>;

    return (
        <div>
            <h1>{user.name}</h1>
            <p>Email: {user.email}</p>
            <p>Joined: {new
Date(user.createdAt).toLocaleDateString()}</p>
        </div>
    );
};

export default UserProfile;
```

5. RESULTS

i. Landing Page

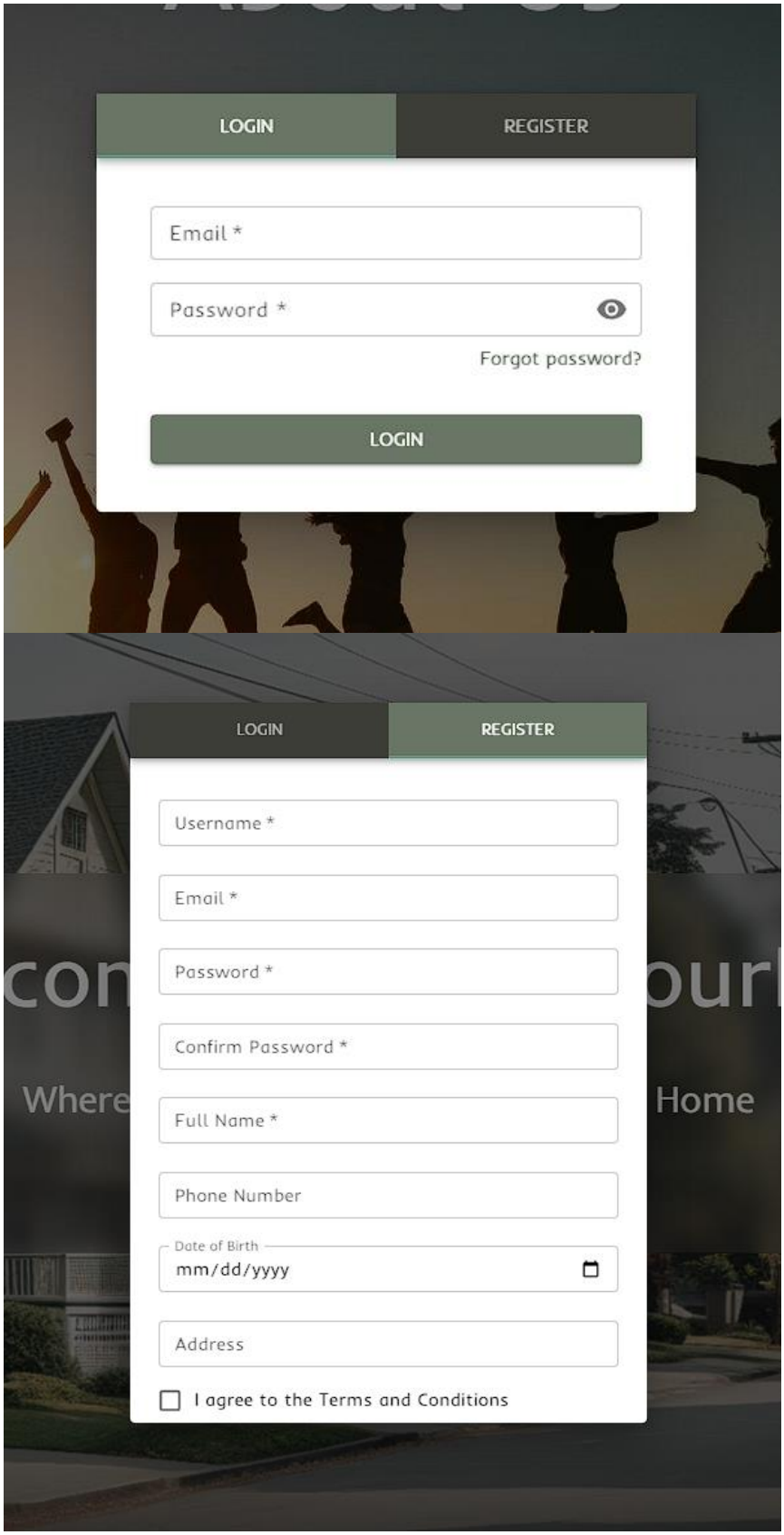
The landing page of NeighborNet is crafted to be both attractive and user-friendly, serving as an engaging introduction to the platform. It features a clean and simple design that highlights the core functionalities of NeighborNet. This section prominently displays the key features of the platform, including user profile management, help requests, and resource sharing. By clearly outlining these functionalities, the landing page effectively communicates the platform's value and purpose to potential users.

The "About Us" section on the landing page provides an overview of NeighborNet's mission, vision, and core values. It is designed to convey the platform's dedication to enhancing community support and interaction in a straightforward manner. This section helps users understand the platform's goals and the impact it aims to make within local neighborhoods.

Additionally, the landing page includes a "Testimonials" area where feedback from current users is showcased. These testimonials offer real-world endorsements of NeighborNet's effectiveness and build credibility by highlighting positive user experiences.

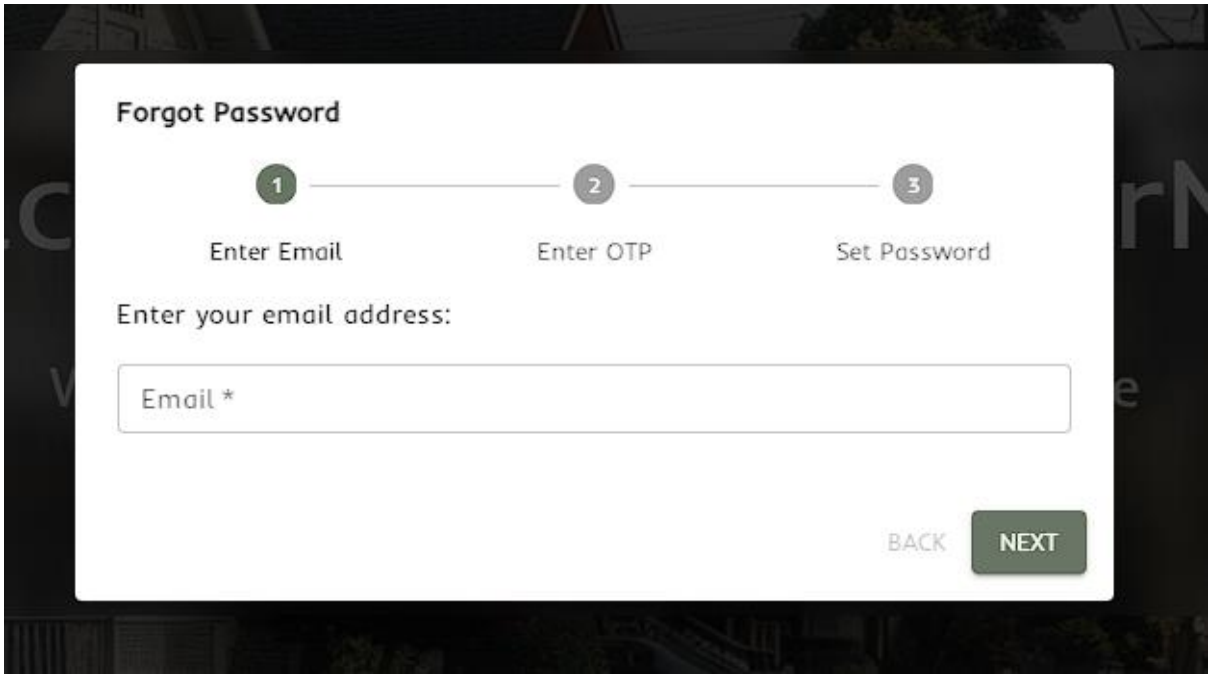
ii. Login And Registration Window

Login and registration dialogues with simple and user friendly design.



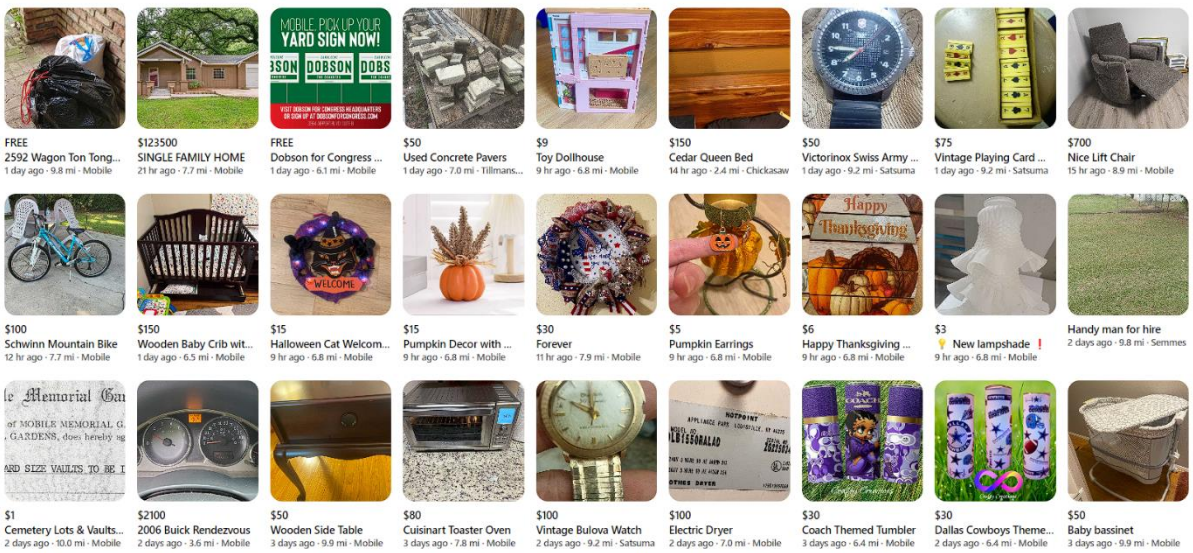
iii. OTP Verification Dialogue

OTP verification dialogue which uses steppers to track the progress until the otpis verified and the required action is performed.



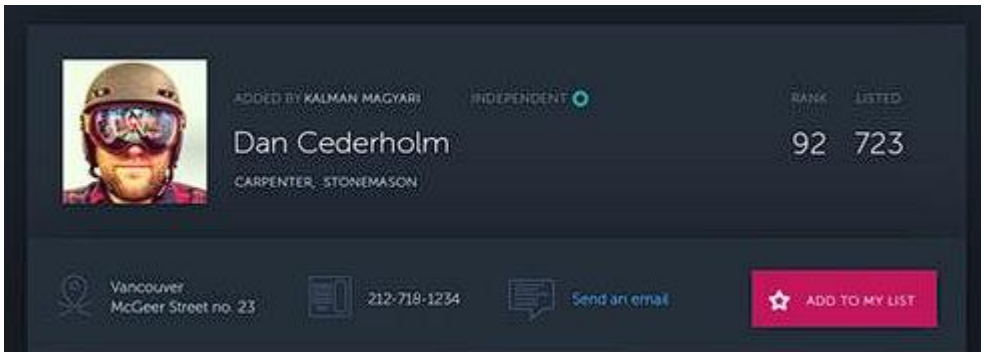
iv. Help Requests

A Page to post and view the help requests uploaded in the platform with location filters



v. User Profile Page

User Profile page that displays the basic details of a registered individual and the allows the user to edit his/her personal details



6. CONCLUSION

NeighborNet represents a significant step forward in fostering community engagement and support through a streamlined and user-friendly web platform. By focusing on core functionalities like user profile management, help requests, and resource sharing, the platform addresses fundamental needs in neighborhood interaction without the complexity found in more feature-rich systems.

The choice to build NeighborNet with React.js, Node.js, Express, and MongoDB ensures a robust and scalable solution, while the incorporation of Material-UI enhances the visual appeal and usability of the interface. The platform's simplicity and ease of use make it accessible to users of all technical backgrounds, fulfilling the goal of a straightforward tool for local engagement.

In contrast to existing systems that may be overly complex or costly, NeighborNet offers a free, secure, and no-frills solution that prioritizes essential features and user experience. Its focus on reliability and practical functionality over additional hype ensures that it meets the real-world needs of its users effectively.

Overall, NeighborNet stands out as a practical and accessible solution for enhancing neighborhood interactions, combining modern technology with a clear commitment to simplicity and community support.

7. FUTURE IMPLICATIONS

As NeighborNet continues to evolve, several future implications and potential developments could further enhance its impact and effectiveness within community interactions and support. These implications include the following:

1. **Enhanced Features and Integrations:**
    - **Advanced Analytics**: Incorporating advanced analytics could provide deeper insights into community engagement patterns and user behavior. This data could help in refining features, improving user experience, and tailoring support to meet community needs more effectively.
    - **Third-Party Integrations**: Integration with other popular services and platforms (such as social media or local event platforms) could enhance the functionality of NeighborNet, providing users with a more interconnected experience. This could include features such as social sharing, event synchronization, or direct messaging with external apps.
2. **Mobile Application Development:**
    - **Dedicated Mobile App**: Developing a dedicated mobile application for NeighborNet could significantly improve accessibility and user engagement. A mobile app could offer notifications, offline access, and a more optimized interface for mobile devices, making it easier for users to interact with the platform on the go.
3. **AI and Machine Learning:**
    - **Personalization**: Implementing AI and machine learning algorithms could enable personalized content and recommendations based on user preferences and behavior. For example, the platform could suggest relevant resources, help requests, or community events tailored to individual users.
    - **Automated Moderation**: AI-driven moderation tools could help maintain a safe and respectful community environment by automatically detecting and managing inappropriate content or behavior.
4. **Community Building and Engagement:**
    - **Enhanced Social Features**: Adding features such as forums, discussion boards, or group chats could further facilitate community building and

interaction. These features would allow users to engage in more meaningful conversations and collaborations within their neighborhoods.

- **Gamification**: Introducing gamification elements, such as badges, leaderboards, or rewards for active participation, could increase user engagement and motivation to contribute to the community.

5. **Scalability and Performance Optimization:**
   - **Infrastructure Improvements**: As the platform grows, investing in scalable infrastructure and performance optimization will be crucial to handle increased user traffic and data. Leveraging cloud-based solutions and optimizing database queries can help ensure that the platform remains responsive and reliable.
6. **Expanded Accessibility and Inclusivity:**
   - **Multilingual Support**: Expanding the platform's language support could make NeighborNet more accessible to diverse communities. Providing multilingual options would help users from different linguistic backgrounds engage more effectively with the platform.
   - **Accessibility Features**: Enhancing accessibility features, such as screen readers or customizable text sizes, could ensure that users with disabilities can fully participate in community interactions.
7. **Privacy and Security Enhancements:**
   - **Advanced Security Measures**: Continually updating security protocols to address emerging threats will be essential for protecting user data and maintaining trust. Implementing advanced encryption, regular security audits, and user data protection measures will help safeguard against potential vulnerabilities.
   - **User Data Control**: Providing users with more control over their data, including options to manage privacy settings and data sharing preferences, could enhance user trust and compliance with data protection regulations.
8. **Community Feedback and Iterative Development:**
   - **User Feedback Integration**: Actively soliciting and incorporating user feedback into the development process can help ensure that NeighborNet evolves in line with user needs and expectations. Regularly updating features based on feedback can help maintain user satisfaction and relevance.

8. REFERENCES

i. “Building Community-Based Systems for Community Development: A Review of Literature”, R. R. Gibb and T. H. Gibb, Community Development Journal

ii. "The Impact of Social Networks on Community Engagement: A Systematic Review", J. G. Young and L. J. Clarke, Social Network Analysis and Mining

iii. "A Survey of Cloud Computing and Its Applications", S. M. Khan and A. R. Khan, International Journal of Cloud Computing and Services Science

iv. "Modern Web Development with React and Node.js", A. Smith and J. Brown, Web Development Journal

v. Material-UI Documentation - https://mui.com/

vi. React.js Documentation - https://legacy.reactjs.org/docs/getting-started.html

vii. Node.js Documentation - https://nodejs.org/en/learn/getting-started/

viii. MongoDB Documentation - https://www.mongodb.com/docs/

ix. Express.js Documentation - https://expressjs.com/