International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

ServiceSetu

Design and Development of a Web-Based Local Service Provider Booking Platform

Mr. Vikas Yadav
Department of Computer Science, Chikitsak Samuha's Patkar Varde College

---

Abstract— People who need a local service such as a plumber, an electrician, a tutor or a digital marketing expert usually depend on word of mouth, pamphlets or a few known phone numbers, and small independent service providers have no low-cost place where customers can find them. This paper describes ServiceSetu, a browser-based platform that connects consumers and service providers in one application. The front-end is a React single-page application that uses React Router, Bootstrap and Axios, the server side is a Node.js and Express REST API, and all records are kept in a MySQL database. A provider registers with the service offered, location, price, company name and experience. A consumer searches for a service, views the matching providers and books an appointment with a single click. Every appointment is stored with its date and time; the consumer can view and cancel bookings, and the provider can view the customers who have booked and reject a booking. The role chosen at login decides which pages and menu are shown. The system follows a three-tier architecture and was built with an incremental model planned over fourteen weeks, and its analysis and design are documented with use case, activity, E-R, data flow, sequence and class diagrams and with flowcharts. Only free and open-source software is used. A test plan covering functional behaviour, role-based access, data consistency, response time and usability is proposed, and the present limitations, such as plain-text password storage and partly non-parameterised queries, are stated openly together with the planned improvements.

Keywords— *Local Service Marketplace, Service Provider Booking, Appointment Management, Three-Tier Architecture, React, Node.js, Express.js, MySQL, Role-Based Navigation, Web Application.*

1. INTRODUCTION

Almost every household and small business needs a local service from time to time: a plumber to repair a pipe, an electrician for wiring work, a tutor, a photographer or a digital marketing expert to promote a small shop. In most towns and cities these professionals are still found through relatives, neighbours, printed pamphlets or a few known phone numbers. A customer cannot easily compare providers, does not know the usual price in advance and keeps no record of whom he or she has booked. A provider, especially an independent worker or a small firm, depends on a small circle of customers because there is no cheap way to show the service, location, price and experience to people who are actually searching for it.

Online platforms that bring two groups of users together are called two-sided platforms, and their value grows with the number of users on each side [1], [2]. Large marketplaces such as Urban Company [3], Justdial [4] and TaskRabbit [5] apply this idea to local services on a very big scale, with ratings, payments and scheduling. They are, however, complex commercial systems that an institution or a small community cannot modify, and their features and fees may be more than a neighbourhood, a housing society or a small agency needs. A simple platform that handles only the essential cycle of publishing a service, finding a provider and booking an appointment is therefore still useful, both for local use and as a clear teaching example of full-stack development.

ServiceSetu (the name comes from *setu*, the Hindi word for bridge) was designed to be such a platform. A React interface talks to a Node.js and Express back-end, MySQL holds consumers, providers and appointments, and two roles, consumer and service provider, see different pages. The work makes the following contributions:

- **One platform for both sides:** consumers and providers register and log in separately and are guided to pages that suit their role.
- **A short booking cycle:** search by service name, a one-click Appoint action, a list of own appointments with cancel, and a provider dashboard with reject.
- **Three-tier open-source implementation:** React, Node.js with Express, and MySQL, with seven JSON endpoints and three tables.
- **Documented design:** use case, activity, E-R, data flow, sequence and class diagrams, a data dictionary and flowcharts of the main procedures.
- **Honest status report:** the known security weaknesses of this version are listed together with a plan to remove them and a plan to evaluate the system.

2. RELATED WORK

2.1 Online Service Marketplaces

Service marketplaces let customers search for a professional, compare options and book a visit online. Urban Company [3], Justdial [4] and TaskRabbit [5] are well-known examples. According to their public descriptions they offer features such as verified profiles, ratings, online payment and scheduling, and they work on a commercial basis in which providers may pay fees or commissions. Their strength is scale; their limitation for a small community is that they are closed hosted services that cannot be adapted to local needs, and the large number of features can make them heavy for a person who only wants to find a plumber nearby.

2.2 Two-Sided Platforms and Reputation

Economic studies of two-sided markets explain that a platform must attract both sides at once, because consumers

come only if providers are present and providers come only if consumers are present [1], [2]. A simple, free and easy-to-use registration for both roles lowers this entry barrier, which is the approach taken in ServiceSetu. Reputation systems, in which users rate one another after a transaction, are a well-studied way to build trust between strangers on such platforms [6]. ServiceSetu does not implement ratings in this version, but its appointment record is the natural basis on which a review feature can be added later.

2.3 Web Architecture and Databases

Modern web applications separate the user interface, the application logic and the data store. The back-end exposes its functions as a REST-style interface that exchanges JSON with the browser [7], and a single-page application built with a library such as React updates the screen without reloading the page [8]. Node.js with the Express framework is a lightweight way to write such an interface in the same language as the front-end [9]. MySQL is a widely used relational database that keeps data in tables linked by keys [10], and sound design follows the relational and normalisation principles described in standard textbooks [11]. ServiceSetu follows this common pattern.

2.4 Web Security

The OWASP Top Ten lists the most frequent web risks, including injection, broken authentication and broken access control [12]. Two of its main recommendations are to use parameterised database queries and to store passwords only as salted hashes, for which the adaptive bcrypt scheme was designed [13]. These guidelines were used in this work to identify the weaknesses of the present version, which are described in Section 7 and are to be removed in the next version.

2.5 Research Gap

The platforms reviewed are strong at large-scale commercial booking but are hosted services that an institution cannot modify, and they carry features and fees that a small community may not need. To the best of the author's knowledge, they do not offer a very small, free and open-source booking system with separate consumer and provider roles that can be installed on a local computer and studied end to end. ServiceSetu targets this gap. Table 1 contrasts the two approaches on the basis of publicly described features; it is a feature comparison and not a performance benchmark.

Table 1. Comparison of large marketplaces with ServiceSetu

Aspect	Large marketplaces [3]–[5]	ServiceSetu (proposed)
Purpose	Commercial marketplace for many cities	Simple local booking platform
Cost to providers	Fees or commission may apply	Free; open-source software
Main features	Ratings, payment, verification, scheduling	Registration, search, booking, view, cancel / reject
Customisation	Not possible for the user	Source code can be modified
Deployment	Hosted by the company	Own computer or college LAN

3. PROPOSED FRAMEWORK

This section introduces the structure of ServiceSetu. Section 4 explains the methods and formulas used, and Section 5 covers the implementation.

3.1 Overview

ServiceSetu consists of a browser interface, a back-end that offers a REST API, and a relational database. A new user chooses the role of consumer or service provider and registers. After login, a consumer types the name of a service and receives a table of the matching providers with their location, price and company, and books an appointment with one click. The server stores the appointment with its date and time. The consumer can open a page that lists his or her appointments and can cancel one; the provider opens a dashboard that lists the customers who have booked and can reject a booking. The database is the single record of users and appointments.

3.2 User Roles

Consumer. Can register, log in, search for a service, view providers, book an appointment, view and cancel his or her own appointments, view the About Us page and log out. **Service provider.** Can register with the service details, log in, view the customers who have booked, reject an appointment, view his or her profile and the About Us page, and log out. The role selected at login decides the pages and the navigation menu that are shown.

3.3 System Components

The platform is organised into the following components:

- **User interface:** a React single-page application with pages for login, sign-up, home (search), booking, consumer profile, provider dashboard and about us.
- **Back-end:** Node.js and Express receive the API calls, run the business logic and use the cors package so that the front-end may call the server.
- **Database layer:** MySQL keeps the tables consumers, users (providers) and appointments.
- **Session handling:** the logged-in user record is kept in the browser's session storage and cleared at logout.

- **Documentation:** the analysis and design are documented with standard UML, E-R and data flow diagrams.

3.4 Operations and Endpoints

The server offers seven endpoints, which are listed in Table 2. All of them receive and return JSON. Each operation that changes data works on one record at a time, so that a booking is either stored completely or not at all.

Table 2. Operations and REST endpoints of ServiceSetu

Operation	User	Endpoint
Register	Both	/signup
Log in	Both	/login
Search providers	Consumer	/getServices
Book appointment	Consumer	/apply
View own bookings	Consumer	/getAllAppoints
View booked customers	Provider	/viewAppoints
Cancel / reject	Both	/delete

An appointment has a very simple life cycle. It is created when a consumer clicks Appoint, from then on it is visible to both the consumer and the provider, and it ends when either side deletes it by cancelling or rejecting it.

4. METHODOLOGY

4.1 Problem Statement

The task is to let providers of local services publish their services and let consumers find them and book them, with a clear and permanent record of every booking. For every request the platform receives a role and either a registration, a login, a search word or an appointment action, and it must return the correct data for that role. The rules must be applied in the same way every time, and the data of one user must not be shown to another user. The method is divided into four stages: registering and logging in, searching providers, booking an appointment, and viewing and cancelling appointments. The complete flow of these stages is drawn as a single flowchart in Figure 2 (Section 5.3).

4.2 Development Model and Schedule

The system was developed with the incremental model, in which a working system is built in small parts and each part is designed, coded, tested and integrated before the next one is added [14]–[16]. The first increment covered the database and the registration and login of both roles. The second added the service search. The third added booking, the consumer's appointment list with cancel, and the provider dashboard with reject. The fourth covered testing with sample data, correction of defects and documentation. The work was planned over fourteen weeks, from 20 July to 25 October 2026. The expected time of each activity was estimated from an optimistic time O, a most likely time M and a pessimistic time P, as in the PERT technique:

$$TE = (O + 4M + P) / 6 \quad (1)$$

For the nine planned activities these estimates give a critical path of topic selection, literature review, analysis, design, development, integration and documentation with a total expected duration of 14 weeks, while testing has one week of slack.

4.3 Searching Providers

Let P be the set of registered providers and $service(p)$ the service name stored for a provider p. When a consumer types a word q, the server returns the set

$$S(q) = \{ p \in P : service(p) \text{ starts with } q \} \quad (2)$$

which is obtained with one query on the users table. The result is sent to the browser as a list containing, for every matching provider, the name, e-mail, service, location, price and company. If $S(q)$ is empty the consumer sees an empty table and can search again.

4.4 Booking an Appointment

When the consumer clicks Appoint for a provider p, the server creates a record

$$A = (id, c, p, s, t, d) \quad (3)$$

where id is generated by the database, c is the identifier of the logged-in consumer, p the identifier of the chosen provider, s the service name, and t and d the time and date of the booking. The record is inserted into the appointments table, and the interface returns to the home page with a confirmation.

4.5 Viewing and Cancelling Appointments

The list of a consumer c is the set of all appointments with customerID equal to c, joined with the provider details, and the list of a provider p is the set of all appointments with userID equal to p, joined with the customer details:

$$A_c = \{ a : a.c = c \}, \quad A_p = \{ a : a.p = p \} \quad (4)$$

When the consumer clicks Delete or the provider clicks Reject, the server removes the record identified by its appointment id and the page is loaded again so that both lists stay consistent with the database.

4.6 Role-Based Navigation

After a successful login the user record, including the role, is saved in the browser's session storage. A consumer is sent to the home page and sees the consumer menu, whereas a provider is sent to the dashboard and sees the provider menu. At logout the stored record is cleared and the login page is shown. In this version the role check is performed mainly by the interface; server-side authorisation with tokens is part of the planned work (Section 7).

5. SYSTEM IMPLEMENTATION

5.1 System Architecture

ServiceSetu follows a three-tier architecture, illustrated in Figure 1. The presentation tier is the React application that runs in the browser. The application tier is the Node.js and Express server, which runs on port 8081 and is called by the front-end running on port 3000. The data tier is the MySQL database named pathkar. The tiers talk to one another through HTTP and JSON between the first two and SQL queries between the last two, so that the interface, the logic and the data can be changed independently.

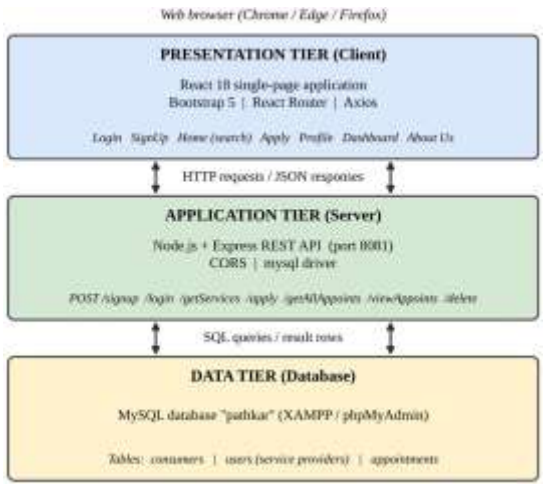


Figure 1. Proposed three-tier architecture of ServiceSetu, showing the React front-end, the Node.js and Express application tier and the MySQL database.

5.2 Technology Stack and Security

Only open-source tools are used, which keeps the software cost at zero and makes the system easy to maintain. The interface is built with React 18, React Router 6 and Bootstrap 5 with React-Bootstrap, and Axios sends the requests to the server. MySQL runs through XAMPP, and phpMyAdmin is used to create the database and import the SQL script. Visual Studio Code and npm are the development tools. Table 3 lists the stack, and the whole system can run on an ordinary computer or on a college LAN.

Table 3. Technology stack of ServiceSetu

Component	Technology used
User interface	React 18, React Router 6, Bootstrap 5, React-Bootstrap
HTTP client	Axios
Server side	Node.js, Express 4, cors, nodemon
Database	MySQL (XAMPP), phpMyAdmin
Session	Browser session storage
Development tools	Visual Studio Code, npm, browser developer tools

With respect to security, the signup and login queries use query placeholders and the role decides the pages shown. In the present version, however, passwords are stored as plain text and some queries are built by joining strings, which the OWASP guidelines identify as risks [12]. The planned remedy is to store salted bcrypt hashes [13], to use parameterised queries for every endpoint, to validate all input and to add token-based authentication.

5.3 Logical Flow of the System

Figure 2 combines the logic of ServiceSetu in one flowchart. The user first signs up or logs in; invalid details return the user to the login step. The user type then decides the path. A consumer enters a service name and the server searches the users table; if no provider is found the consumer searches again, otherwise the consumer clicks Appoint, the appointment is saved and the list of the consumer's appointments is shown, from which an appointment can be cancelled. A provider opens the dashboard, where the customers who have booked are listed, and may reject a booking. A cancelled or rejected appointment is deleted and the list is reloaded. Finally the user logs out and the session is cleared.

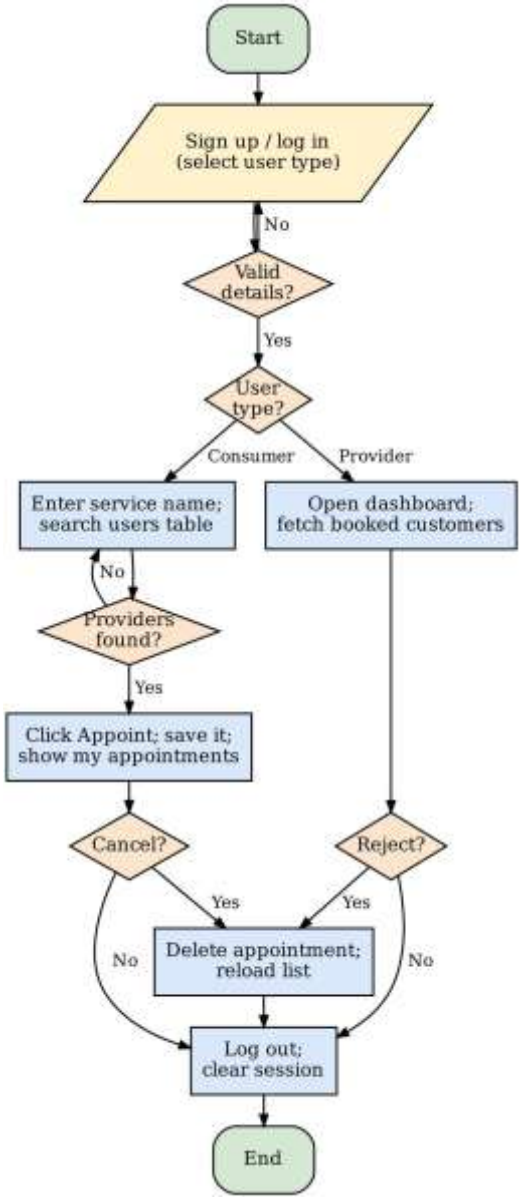


Figure 2. Procedural flow of ServiceSetu, from sign-up or login to logout.

5.4 Database Design

A relational database suits ServiceSetu because its data is well structured and strongly linked. The schema is modelled with an entity-relationship diagram (Figure 3) and with UML diagrams [17]. The consumers table stores each customer, the users table stores each service provider with the service, location, price, company name and experience, and the appointments table stores each booking with the identifiers of the provider and the consumer, the service, the time and the date. A consumer can have many appointments and a provider can receive many appointments, while every appointment belongs to exactly one consumer and one provider. Each table describes a single entity and every non-key column depends on its primary key, in line with the

normalisation principles in [11]. The e-mail address of a consumer is unique. In this version the link between an appointment and its two users is kept by the application; formal foreign-key constraints and indexes on frequently searched columns such as the service name are planned.

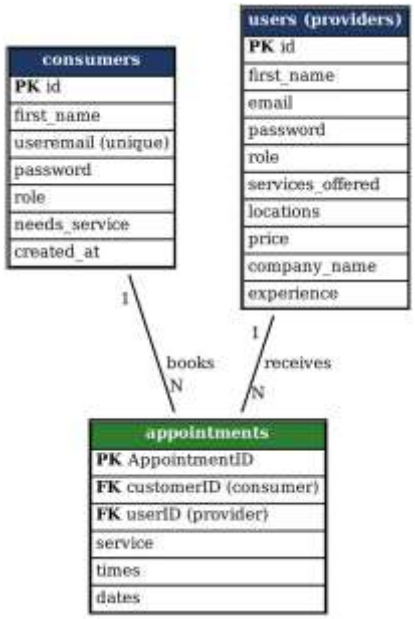


Figure 3. Entity-relationship diagram of ServiceSetu.

6. EVALUATION METHODOLOGY

The system is to be assessed through functional, access, consistency and usability testing in order to confirm that every function works for the correct role and that the stored data stays correct. The test set includes normal use, empty and wrong inputs, duplicate registration, wrong credentials and actions attempted by the wrong role, so that both expected and unexpected situations are covered.

6.1 Metrics

- **Functional correctness:** share of the planned test cases in which the observed result equals the expected result.
- **Role separation:** whether the pages and actions of one role are hidden from the other role.
- **Data consistency:** whether the lists shown to consumers and providers agree with the rows stored in the appointments table after booking, cancelling and rejecting.
- **Response time:** time taken by search and booking requests on a local network, with the target of about one second.
- **Input robustness:** behaviour for empty search words, wrong credentials, duplicate e-mail addresses and a stopped back-end.
- **Usability:** ease of completing the main tasks, judged from the observations and a short questionnaire given to a small group of volunteer users.

6.2 Test Scenarios

A set of scenarios has been prepared for the main features: registration of a consumer and of a provider, login with correct and wrong details, search with matching and non-matching words, booking, viewing the consumer's appointments, cancellation, viewing the provider's dashboard, rejection and logout. Each scenario is given an expected outcome derived from the rules of the system. For example, after a consumer books a provider, the appointment must appear in both the consumer's list and the provider's dashboard, and after it is deleted it must disappear from both. Further cases check the interface in several browsers and on a phone-sized screen. Observed results will be compared with the expected outcomes, and all reported figures will come from real test runs.

6.3 Security Checks

As a baseline, the endpoints will also be tested with unexpected and malicious input, such as quotation marks and SQL keywords in the form fields, in order to document the weaknesses of the present version in Section 7 and to confirm later that the planned fixes remove them.

7. LIMITATIONS AND FUTURE WORK

7.1 Limitations

The present version is a research and learning prototype. Passwords are stored as plain text, some queries are not parameterised, and there is no token-based authentication, so role checks are made mainly by the interface. An appointment stores only the time and date of booking: the consumer cannot choose a time slot, and the provider cannot accept or confirm a booking. The search matches only the beginning of the service name and does not filter by location, price or experience. There are no ratings, no payments, no notifications and no administrator panel, and the system has been run only on a single computer with MySQL under XAMPP. Its behaviour under heavy load and its security therefore still need thorough testing.

7.2 Future Work

Planned extensions begin with security: bcrypt password hashing [13], parameterised queries, input validation and token-based authentication. The appointment can then be extended with statuses such as pending, accepted, rejected and completed, a chosen time slot and the availability of the provider. Ratings and reviews [6], search filters for location, price and experience, e-mail or SMS notifications, online payment, real-time chat and an administrator panel are also planned, followed by native Android and iOS applications and deployment on a cloud server with HTTPS and formal foreign keys in the database.

8. CONCLUSION

This paper has presented ServiceSetu, a web-based platform that connects consumers with local service providers. Built with React, Node.js, Express.js and MySQL, it lets providers publish their services, lets consumers search for them and book appointments with a single click, and gives both sides a clear record of every booking. A three-tier design, an incremental development plan and a complete set of design diagrams make the system easy to understand and extend, and the entire stack is free and open source. Since the system is still a prototype with known security weaknesses, no claims about performance, scalability or user benefit are made until the planned evaluation has been

carried out; the security improvements, the user trials and the extensions listed above form the next stage of this work.

ACKNOWLEDGMENT

The author sincerely thanks the project guide, Ms. Ahinsa Gaikwad, for her guidance and encouragement throughout this work, and Ms. Karishma Jain, project coordinator, for her support. The author is also grateful to the Department of Computer Science and the college management for providing the facilities needed to carry out this research, and to classmates and local service providers for their feedback.

REFERENCES

[1] J.-C. Rochet and J. Tirole, "Platform competition in two-sided markets," *J. Eur. Econ. Assoc.*, vol. 1, no. 4, pp. 990–1029, 2003.

[2] G. G. Parker, M. W. Van Alstyne and S. P. Choudary, *Platform Revolution: How Networked Markets Are Transforming the Economy and How to Make Them Work for You*. New York, NY, USA: W. W. Norton, 2016.

[3] Urban Company. [Online]. Available: https://www.urbancompany.com

[4] Justdial. [Online]. Available: https://www.justdial.com

[5] TaskRabbit. [Online]. Available: https://www.taskrabbit.com

[6] P. Resnick, R. Zeckhauser, E. Friedman and K. Kuwabara, "Reputation systems," *Commun. ACM*, vol. 43, no. 12, pp. 45–48, 2000.

[7] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. of California, Irvine, CA, USA, 2000. [Online]. Available: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm

[8] React Documentation. [Online]. Available: https://react.dev

[9] Node.js Documentation and Express Documentation. [Online]. Available: https://nodejs.org and https://expressjs.com

[10] MySQL Reference Manual. [Online]. Available: https://dev.mysql.com/doc

[11] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Harlow, U.K.: Pearson Education, 2016.

[12] OWASP Foundation, "OWASP Top Ten Web Application Security Risks." [Online]. Available: https://owasp.org/www-project-top-ten/

[13] N. Provos and D. Mazières, "A future-adaptable password scheme," in *Proc. USENIX Annu. Tech. Conf.*, Monterey, CA, USA, 1999, pp. 81–91.

[14] C. Larman and V. R. Basili, "Iterative and incremental developments: a brief history," *Computer*, vol. 36, no. 6, pp. 47–56, Jun. 2003.

[15] I. Sommerville, *Software Engineering*, 10th ed. Harlow, U.K.: Pearson Education, 2016.

[16] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. New York, NY, USA: McGraw-Hill Education, 2019.

[17] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2003.