Project Proposal

QuickClean - Simplified Maid Booking Platform

Section - [18]

Group 11

Group Member

[22101262 - Sayed Ilham Azhar Harun]

QuickClean - Simplified Maid Booking Platform

QuickClean is a streamlined web-based platform designed to connect customers with domestic cleaning service providers (maids). Unlike complex marketplace solutions, QuickClean focuses on simplicity and ease of use, providing essential booking functionality without overwhelming features. The name "QuickClean" emphasizes the platform's core value proposition: fast, straightforward access to cleaning services without complicated processes.

Project Rationale

Background

The domestic service industry faces a significant accessibility challenge. While there is clear demand for cleaning services, many existing platforms are either too complex for casual users or lack basic digital infrastructure. Small-scale service providers often struggle to manage bookings efficiently, while customers face difficulties in finding reliable services.

QuickClean addresses this gap by providing a lightweight, easy-to-use platform that handles the essentials: user registration, service browsing, and booking management. By focusing on core functionality rather than feature bloat, the platform serves users who need a straightforward solution without the learning curve of enterprise-level systems.

Scope

Target Audience:

- **Primary Users:** Urban households and individuals seeking occasional or regular domestic cleaning services
- **Service Providers:** Independent maids and small cleaning service operators looking for a simple digital presence
- **Administrators:** Platform managers who oversee service quality and user verification

User Benefits:

- Customers gain easy access to verified service providers with transparent pricing
- Maids receive a professional platform to showcase their services and manage bookings
- Admins maintain quality control through verification and category management

Estimated Reach: Initially targeting small to medium-sized urban areas with 500-1000 active users in the first phase, scalable to larger markets.

Project Objectives

1. Develop a secure user authentication system with email verification to ensure genuine user accounts
2. Create an intuitive admin panel for managing service categories and verifying service providers
3. Implement a straightforward booking system that allows customers to request services easily
4. Establish automated email notifications to keep users informed about booking status
5. Build a clean, responsive interface that works seamlessly across desktop and mobile devices

Project Approach

Technology Stack

Frontend:

- React.js - Component-based UI development
- TailwindCSS - Modern, responsive styling

Backend:

- Node.js with Express.js - RESTful API development
- MongoDB with Mongoose - Flexible data storage

Third-Party Services:

- SendGrid API - Email verification and notifications
- Vercel - Deployment and hosting

Risk Assessment

Technical Risks:

- **Email Deliverability:** Mitigation through SendGrid configuration and spam folder testing
- **Data Security:** Implement JWT authentication and password hashing (bcrypt)
- **Database Performance:** Use proper indexing and pagination for maid listings

Project Risks:

- **Scope Creep:** Maintain strict feature boundaries; defer advanced features to future versions
- **Time Management:** Weekly milestones with buffer time for testing and debugging
- **API Dependency:** Have fallback mechanisms if SendGrid experiences downtime

Requirements

-

Requirement No.	Requirement
1	As a **Customer/Maid**, I want to be able to **register on the platform and verify my email address** so that I can securely access the system.
	Feature 1: User Registration with Role-Based Account Creation The system will implement a User Registration API endpoint (POST /api/auth/register) that accepts user data including name (string, required), email (string, required, unique), phone number (string, required), role (enum: "Customer" or "Maid"), and password (string, minimum 8 characters, required). The backend will validate all inputs, check for existing email in the User collection, hash the password using bcrypt with salt rounds of 10, and generate a unique verification token (UUID v4). The User document will be created in MongoDB with fields: _id, name, email, phone, role, passwordHash, verificationToken, isVerified (default: false), createdAt, and updatedAt. The API will return a success response (201 status) with the user ID and a message indicating verification email has been sent, or return validation errors (400 status) if data is invalid or email already exists.

	Feature 2: Automated Email Verification via SendGrid API Upon successful user registration, the system will trigger an asynchronous email service function that integrates with SendGrid API v3. The function will construct a verification email containing the user's name and a clickable verification link in the format: https://quickclean.vercel.app/verify-email?token={verificationToken}. The email template will include QuickClean branding, a clear call-to-action button, and instructions. SendGrid API will be called with the recipient's email, subject line ("Verify Your QuickClean Account"), HTML body, and sender address (noreply@quickclean.com). The system will log the SendGrid response and handle failures gracefully by storing failed email attempts in an EmailLog collection for retry logic. The verification token will remain valid indefinitely until used or the user requests a new one.
	Feature 3:Account Restriction Based on Verification Status The system will implement a middleware function (requireVerifiedUser) that checks the isVerified field of the authenticated user's document before allowing access to protected routes. When a verified user attempts to log in (before clicking the verification link), the login will succeed and return a JWT token, but any subsequent API calls to protected endpoints (such as /api/bookings, /api/profile/update, or /api/maids/search) will be intercepted by the middleware. If isVerified is false, the middleware will return a 403 Forbidden

	response with the message "Please verify your email address to access this feature" and include a link to resend the verification email (POST /api/auth/resend-verification). The user's dashboard will display a prominent banner alert stating "Your account is not verified. Please check your email." This ensures unverified users can log in to see their account status but cannot perform any core actions.
	Feature 4: Secure Login with JWT Token Generation The system will implement a Login API endpoint (POST /api/auth/login) that accepts email and password. The backend will query the User collection to find a document matching the email, return a 401 error if no user is found, and use bcrypt's compare function to validate the provided password against the stored passwordHash. Upon successful authentication, the system will generate a JSON Web Token (JWT) using the jsonwebtoken library, signing it with a secret key stored in environment variables. The JWT payload will include userId, email, role, and isVerified, with an expiration time of 7 days. The token will be returned in the response body along with user information (excluding sensitive fields like passwordHash). The frontend will store this token in browser localStorage and include it in the Authorization header (Bearer {token}) for all subsequent API requests. The backend will use a verifyToken middleware to decode and validate the JWT on protected

	routes, extracting the user information and attaching it to the request object for use in route handlers.
2	As an **Administrator**, I want to be able to **manage service categories and verify maids** so that the platform offers organized services with quality control.
	Feature 1:Complete CRUD Operations for Service Categories The system will implement a ServiceCategory model in MongoDB with fields: _id, name (string, required, unique), description (string, required), basePrice (number, required, minimum 0), priceUnit (string, enum: ["per hour", "per visit", "per square foot"]), isActive (boolean, default: true), createdAt, and updatedAt. Admin-only API endpoints will be created: POST /api/admin/categories to create new categories with input validation ensuring name uniqueness and positive pricing; GET /api/admin/categories to retrieve all categories with optional filtering by active status and pagination support (default 20 per page); PUT /api/admin/categories/:id to update category details with validation checks; and DELETE /api/admin/categories/:id to soft-delete categories by setting isActive to false rather than removing the document (to maintain data integrity for historical bookings). An admin middleware (requireAdmin) will verify that the authenticated user's role is "Admin" before allowing access to these endpoints. The frontend admin panel will display categories in a table with inline editing capabilities,

	search functionality, and confirmation dialogs for delete operations.
	Feature 2:Maid Profile Submission with Document Upload The system will extend the User model to include maid-specific fields when role is "Maid": profile (embedded object containing: experience (number, years), skills (array of strings), bio (string, max 500 characters), availableServices (array of ServiceCategory IDs referencing the ServiceCategory collection), and hourlyRate (number, minimum 0)), documents (embedded object containing: idType (string, enum: ["NID", "Passport", "Driving License"]), idNumber (string), idDocumentUrl (string, file path or cloud storage URL), uploadedAt (timestamp)), verificationStatus (string, enum: ["Pending", "Approved", "Rejected"], default: "Pending"), and adminNotes (string for rejection reasons or approval comments). During the registration process, if the user selects role "Maid", the frontend will display additional form fields for profile information and a file upload component for ID documents. The backend registration endpoint will handle multipart/form-data, validate the uploaded file (accept only PDF, JPG, PNG formats, max 5MB size), store the file in a /uploads/documents/ directory with a unique filename (using UUID), and save the file path in the idDocumentUrl field. Maids with "Pending" verification status will have limited access until approved.

	Feature 3:Admin Dashboard for Maid Verification Workflow The system will provide an admin dashboard to review and approve/reject maid profiles based on document verificationThe system will implement a dedicated Admin Dashboard API endpoint (GET /api/admin/maids/pending) that retrieves all User documents where role is "Maid" and verificationStatus is "Pending", sorted by createdAt (oldest first). The response will include maid profile details and document URLs. The admin frontend dashboard will display these pending maids in a card-based layout, showing profile picture, name, experience, skills, available services, and a preview/download link for the ID document. Each card will have two action buttons: "Approve" and "Reject". Clicking "Approve" will trigger PUT /api/admin/maids/:id/approve, which updates the maid's verificationStatus to "Approved" and isVerified to true, enabling full platform access. Clicking "Reject" will open a modal requiring the admin to enter a rejection reason, then call PUT /api/admin/maids/:id/reject with the reason, updating verificationStatus to "Rejected" and storing the reason in adminNotes. The dashboard will also display summary statistics: total pending maids, total approved maids, and total rejected maids using aggregation queries.
	Feature 4: Automated Email Notifications for Maid Approval Status The system will implement an email notification service that triggers automatically when an admin approves or rejects a maid profile. After

	the approval API (PUT /api/admin/maids/:id/approve) successfully updates the database, an asynchronous function will call SendGrid API to send an approval email to the maid's registered email address. The approval email will include a congratulatory message, confirmation of verification, and a call-to-action button linking to their dashboard to start accepting bookings. Similarly, after rejection (PUT /api/admin/maids/:id/reject), a rejection email will be sent including a polite explanation, the admin's notes on the rejection reason, and instructions to re-submit documents or contact support. Both email templates will be stored as reusable HTML templates with dynamic variables for personalization (maid name, admin notes). The system will log all sent emails in an EmailLog collection with fields: userId, emailType (enum: "Verification", "Approval", "Rejection", "Booking"), status (enum: "Sent", "Failed"), sentAt, and errorMessage (if failed). A background job will retry failed emails every 30 minutes for up to 3 attempts.
3	As a **Customer**, I want to be able to **browse available maids and submit booking requests** so that I can schedule cleaning services.
	Feature 1: Maid Listing with Detailed Profile Display The system will implement a public-facing API endpoint (GET /api/maids) that retrieves all User documents where role is "Maid", verificationStatus is "Approved", and isVerified is true.

	The query will use MongoDB's populate method to expand the availableServices field, replacing ServiceCategory IDs with full category objects containing name, description, and pricing. The response will include: maid's name, profile picture URL (default placeholder if not uploaded), experience in years, bio, skills array, available services with pricing, hourly rate, and average rating (default 0 if no ratings exist). The API will support pagination with query parameters page (default 1) and limit (default 12), returning metadata like totalMaids, currentPage, totalPages, and hasNextPage. The frontend will render maids in a responsive grid layout using cards, each displaying profile picture, name, experience badge, top 3 skills, and a "View Profile" button. Clicking "View Profile" will navigate to /maids/:id route, fetching detailed information via GET /api/maids/:id and displaying full bio, all services with pricing breakdown, customer reviews (if implemented later), and a prominent "Book Now" button.
	Feature 2: Booking Request Submission with Date/Time Selection The system will implement a Booking model in MongoDB with fields: _id, customerId (reference to User collection), maidId (reference to User collection), serviceId (reference to ServiceCategory collection), bookingDate (date, required), bookingTime (string, format "HH:MM", required), duration (number, hours, default 2), address (embedded object with: street, city, postalCode), specialInstructions (string, optional, max 1000 characters), status (string, enum: ["Pending",

	"Accepted", "Rejected", "Completed", "Cancelled"], default: "Pending"), totalPrice (number, calculated based on service basePrice and duration), createdAt, and updatedAt. The booking submission endpoint (POST /api/bookings) will be protected by authentication middleware, requiring a valid JWT token. The request body will be validated to ensure bookingDate is not in the past and bookingTime is within reasonable hours (7 AM - 9 PM). The system will calculate totalPrice by multiplying the selected service's basePrice by duration. The frontend booking form will be accessible from the maid's profile page, featuring a date picker (using a React date library), time slot dropdown (30-minute intervals), service selection dropdown (populated from maid's available services), duration slider (1-8 hours), address input fields auto-filled from customer's profile, and a text area for special instructions. Upon form submission, the booking will be created with "Pending" status.
	Feature 3: Real-time Email Notifications for Booking Events The system will implement an event-driven email notification system triggered by booking-related actions. When a booking is successfully created (POST /api/bookings), two emails will be sent asynchronously via SendGrid API: (1) A confirmation email to the customer containing booking ID, maid name and

	contact, service details, date/time, address, total price, and status ("Pending Confirmation"). The email will include a message explaining that the maid will confirm availability shortly. (2) A new booking alert email to the maid containing customer name and contact, service requested, preferred date/time, address, special instructions, total earnings, and action buttons linking to their dashboard to "Accept" or "Reject" the booking. Both emails will use responsive HTML templates with QuickClean branding. Email sending will be abstracted into a reusable sendEmail utility function accepting parameters: recipient email, template type, and dynamic data object. Failed email sends will be logged in the EmailLog collection and retried using a background job scheduler (implemented with Node-cron), attempting resends every 15 minutes for up to 5 attempts before marking as permanently failed.
	Feature 4: Maid Dashboard for Booking Request Management The system will implement a Maid Dashboard API endpoint (GET /api/maids/bookings) that retrieves all Booking documents where maidId matches the authenticated maid's user ID, sorted by createdAt (newest first). The endpoint will support filtering by status query parameter (e.g., ?status=Pending to show only pending requests). The response will populate customer details (name, phone, profile picture) and service details (name, basePrice). The frontend maid dashboard will display bookings in a tabbed interface with tabs for "Pending" (default), "Accepted", "Completed", and "Rejected". Each booking card

	will show customer information, service details, date/time, address, special instructions, and total earnings. For "Pending" bookings, the card will include two action buttons: "Accept" triggering PUT /api/bookings/:id/accept and "Reject" triggering PUT /api/bookings/:id/reject (with optional rejection reason). The accept endpoint will update the booking's status to "Accepted" and send a confirmation email to the customer stating the maid has confirmed availability, with booking details and a reminder to prepare for the service. The reject endpoint will update status to "Rejected", store the optional reason in a new rejectionReason field, and send an apologetic email to the customer explaining the maid is unavailable, suggesting they browse other maids. All status updates will be transactional to ensure data consistency.

Conclusion

QuickClean represents a focused, achievable solution to the domestic service booking challenge. By concentrating on three essential modules—secure authentication with email verification, admin-controlled service and maid management, and a simple booking workflow;the project delivers genuine value without unnecessary complexity.

The technology stack is modern yet manageable for solo development, with only one third-party API (SendGrid) to reduce integration complexity. This project demonstrates practical full-stack development skills including user authentication, CRUD operations, API integration, and responsive UI design while maintaining a realistic scope for individual completion within academic timelines.