

Apr2024

Dern Support

Prepared by: Belal Moustafa,
Full Stack Developer

Introduction

Congratulations on securing the role of a junior developer at Soume Computing! As part of your responsibilities, you've been tasked with designing and developing a software solution for Dern-Support, a small IT technical support company. The goal is to enhance their business operations and support their expansion of services. This documentation outlines the proposed solution, requirements, design, implementation details, and testing approach for the project.

Business Context and Problem Summary

Dern-Support is an IT technical support company specializing in repairing computer systems for businesses and individuals. Their operations include on-site support for businesses and service delivery options for individuals.

The primary challenge faced by Dern-Support is the need for a software solution that streamlines their operations, manages service requests efficiently, and supports customer interactions seamlessly.

Proposed Solution

The proposed solution is a full-stack software application designed to automate and optimize Dern-Support's business processes. It includes modules for user management, service requests, categorizing services, managing categories, and administrative tasks.

Functional and Non-functional Requirements Specifications

- **Functional Requirements:**
 - **User Management:** Ability to register, login, edit profile, and logout.
 - **Service Management:** CRUD operations for services, categorization, and viewing requests.
 - **Service Requests:** Ability to create, view, accept, and reject service requests.
 - **Admin Functionality:** Access to admin dashboard, managing service requests, and user roles.
- **Non-functional Requirements**
 - **Performance:** Response time for operations, system scalability.
 - **Security:** User authentication, role-based access (admin, user), unique admin code for registration.
 - **Usability:** Intuitive user interfaces, error handling, user feedback mechanisms.
 - **Accessibility:** Alternative layouts, support for screen readers, keyboard navigation.
 - **Compatibility:** Cross-browser compatibility, responsive design for mobile devices.
 - **Data Integrity:** Unique email validation, data encryption for sensitive information.
 - **Efficiency:** Optimized algorithms for data retrieval and processing.

Software Design Documents:

Algorithm Design Documentation

- User Authentication Algorithm:
 - Input: Email, Password, Role (Admin/User), Admin Code (if role is admin).
 - Steps:
 - i. Validate email format and uniqueness.
 - ii. Hash the password securely.
 - iii. Check role and validate admin code if role is admin.
 - iv. Authenticate user credentials against database records.
 - v. Grant access based on role and validity of admin code.
- Service Request Management Algorithm:
 - Input: Service Details, User ID, Request Status.
 - Steps:
 - i. Validate service details (name, description, price, duration).
 - ii. Create or update service record in the database.
 - iii. Process service request based on request status (accepted, waiting, rejected).
 - iv. Update service request status accordingly.
 - v. Notify users/admins about request status changes.

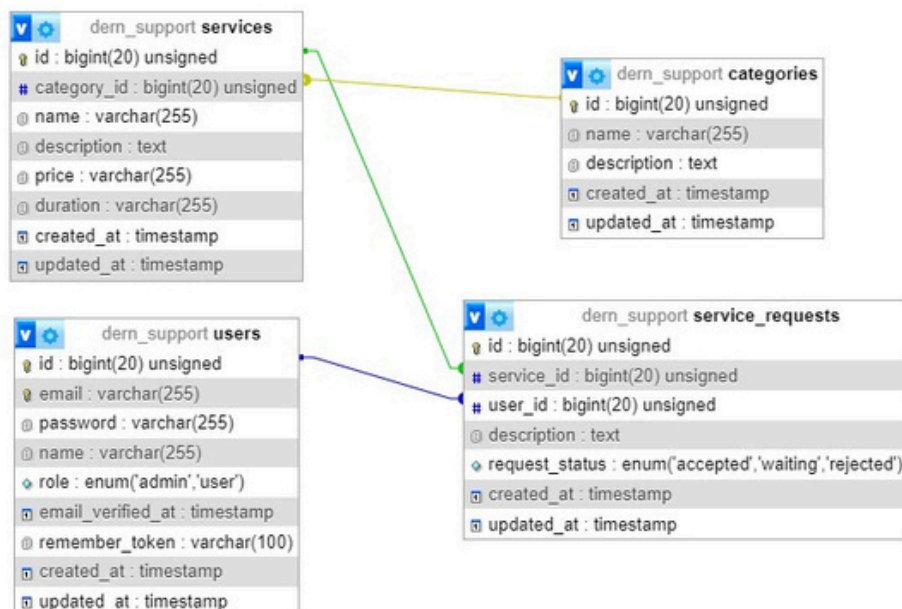
Software Design Documents:

Algorithm Design Documentation

- Mockups and UI Designs:
 - Login/Register Pages: Form fields for email, password, role, and admin code.
 - Admin Dashboard: Overview of service requests, user management options.
 - User Dashboard: View/edit profile, create/view service requests.
 - Service Management: CRUD operations for services, categorization options.
 - Responsive Design: Adaptation for various screen sizes, mobile-friendly layouts.

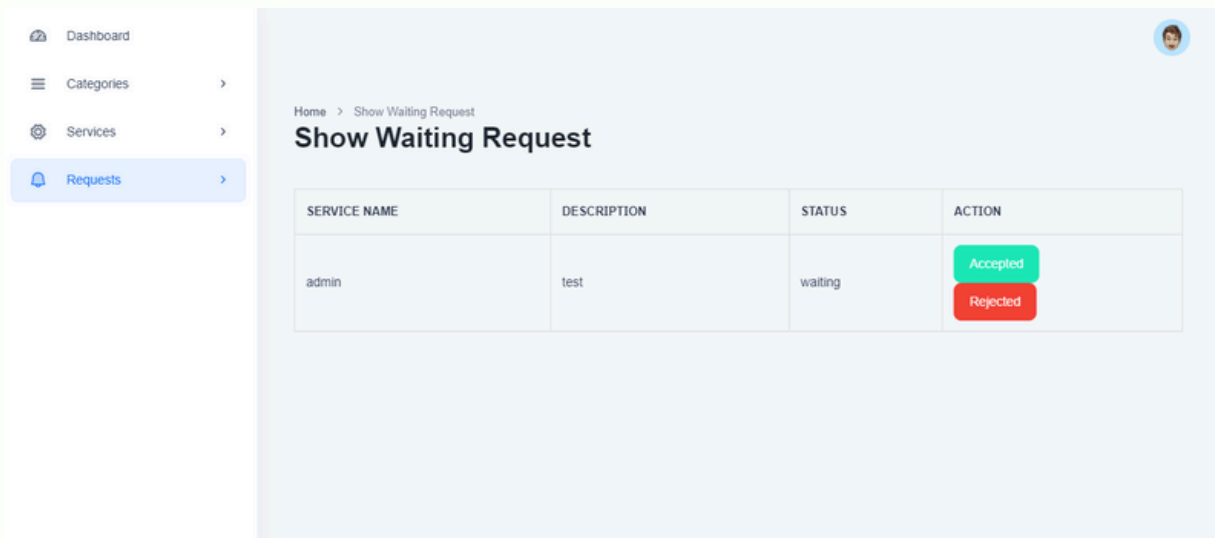
Data Requirements Design

- Entity Relationship Diagram (ERD):
 - Users Table: ID, Email, Password, Name, Role, Admin Code.
 - Categories Table: ID, Name, Description.
 - Services Table: ID, Category ID, Name, Description, Price, Duration.
 - Service Requests Table: ID, Service ID, User ID, Description, Request Status.



Data Requirements Design

- Data Dictionaries:
 - Define data types, constraints, relationships between entities.
 - Example: User Role (Enum: Admin, User), Request Status (Enum: Accepted, Waiting, Rejected).



Component Reuse/Refactoring

- **Reusable Components:**
 - Authentication Middleware: Reusable for user and admin authentication.
 - Service Management Modules: CRUD operations can be reused across different parts of the application.
 - Error Handling Utilities: Centralized error handling functions for consistent user experience.
- **Third-party Integrations:**
 - Laravel Sanctum: API token generation for user authentication.
 - Laravel Livewire: Real-time interaction components for enhanced user experience.

Algorithm and MVC Implementation:

*The software utilizes a structured MVC (Model-View-Controller) architecture implemented in Laravel, a PHP framework. When an admin logs in, the system checks the provided credentials against the database, ensuring the email is unique, the password is securely hashed, and the role is either 'admin' or 'user'. If the role is 'admin', the system validates the provided admin code ('**admin_code_belal**') to grant admin privileges.*

The MVC architecture separates concerns effectively:

Model:	Represents data structures (e.g., User, Service) and interacts with the database.
View	Handles presentation logic, displaying UI elements based on user roles and request status.
Controller	Contains business logic, processing user input, and coordinating interactions between models and views.

Key Performance Indicators (KPIs):

1

Response Time for Service Requests: Monitoring the time taken to process service requests from submission to resolution. Target: <24 hours for non-critical requests.

2

User Satisfaction Ratings: Conducting surveys or feedback mechanisms to assess user satisfaction with the platform's usability, features, and responsiveness. Target: >80% user satisfaction rate.

3

System Uptime: Tracking the system's availability and uptime to ensure uninterrupted service access. Target: >99.9% uptime per month.

4

Adherence to SLAs: Evaluating the system's performance against predefined SLAs, such as response time commitments and service availability guarantees. Target: Meeting or exceeding SLA benchmarks consistently.

Risks and Implications

1

Data Security Risks: Potential risks of data breaches, unauthorized access, or data loss due to vulnerabilities in the system's security measures. **Mitigation:** Implement robust encryption, access controls, and regular security audits.

2

Implementation Complexities: Challenges in integrating various components, ensuring compatibility, and managing dependencies during system implementation. **Mitigation:** Conduct thorough testing, use version control, and employ experienced developers.

3

User Adoption Challenges: Resistance or difficulties in user acceptance of the new system, requiring effective training, user-friendly interfaces, and clear communication of benefits. **Mitigation:** Conduct user training sessions, provide user guides, and gather continuous feedback for improvement.

4

Integration Issues: Potential difficulties in integrating the solution with existing systems, third-party services, or APIs, leading to data inconsistencies or functionality gaps. **Mitigation:** Conduct comprehensive integration testing, use standardized protocols, and establish clear communication with third-party providers.

SDLC Implementation:

1

Requirements Gathering: Gathered business requirements, user stories, and stakeholder inputs to define project scope and objectives.

2

Design: Created software design documents including ER diagrams, UI mockups, algorithm designs, and data dictionaries.

3

Implementation: Developed the solution using Laravel framework, following MVC architecture, and integrating necessary components and functionalities.

4

Testing

- **Testing was conducted at various levels:**
 - Unit testing using PHPUnit for controllers and models.
 - Functional testing for user flows and features using Laravel's testing framework.

```
PS D:\dern-supoort> php artisan test
```

```
WARN Your XML configuration validates against a deprecated schema. Migrate your XML configuration using "--migrate-configuration"!
```

```
PASS Tests\Unit\ExampleTest
✓ that true is true
```

```
PASS Tests\Feature\ExampleTest
✓ the application returns a successful response
```

0.16s

```
PASS Tests\Feature\ServiceControllerTest
✓ search
✓ store
```

0.44s

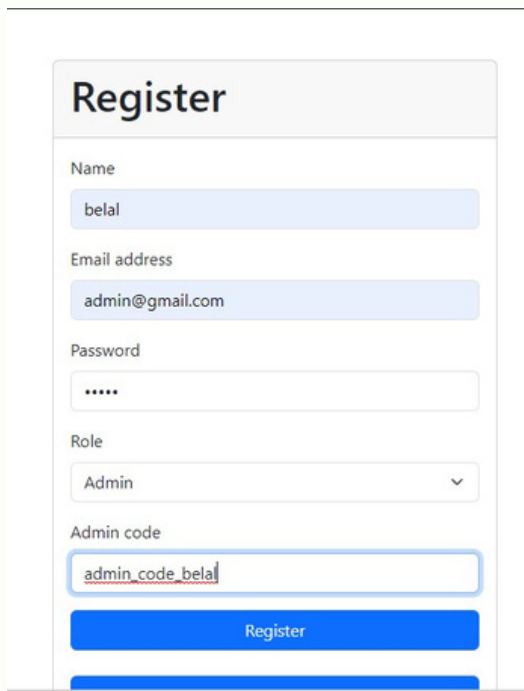
0.05s

```
Tests: 4 passed (9 assertions)
Duration: 1.01s
```

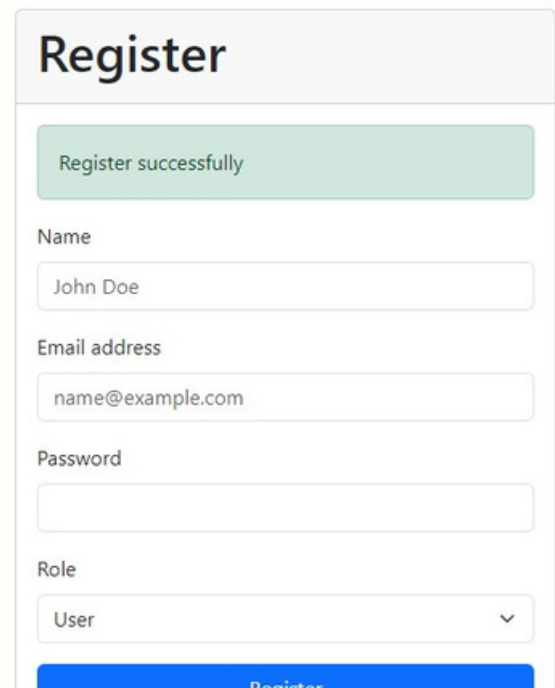
Testing

Testing was conducted at various levels:

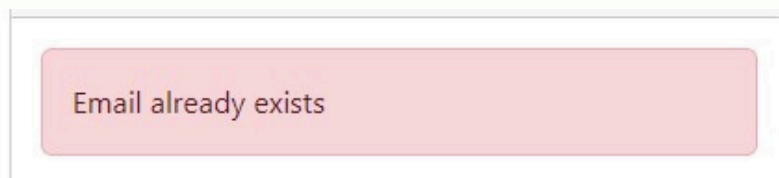
- User acceptance testing (manual) to validate user roles, custom admin codes, and unique email validation during account creation.



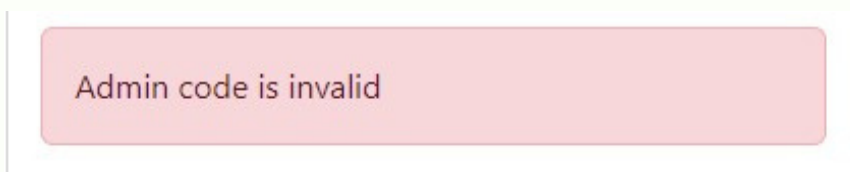
The image shows a 'Register' form with a light gray header. Below the header, there is a green success message box that says 'Register successfully'. The form fields are: Name (John Doe), Email address (name@example.com), Password (empty), Role (User), and Admin code (empty). A blue 'Register' button is at the bottom.



The image shows a 'Register' form with a light gray header. Below the header, there is a red error message box that says 'Email already exists'. The form fields are: Name (John Doe), Email address (name@example.com), Password (empty), Role (User), and Admin code (empty). A blue 'Register' button is at the bottom.



Email already exists



Admin code is invalid

Implementation Details

- Controllers: AuthController, CategoryController, ServiceController, DashboardController, etc.
- Models: User, Category, Service, ServiceRequest, etc.
- Migrations: Defined database schema for users, categories, services, service requests.
- Routes: Defined routes for various functionalities including user authentication, service management, and admin tasks.

Evaluation of the Solution:

The software solution was evaluated based on efficiency, effectiveness, adherence to requirements, user feedback, and refinement iterations. Feedback from stakeholders and users was gathered, analyzed, and used to improve the solution iteratively.

Conclusion

In conclusion, the comprehensive documentation and implementation of the full-stack solution for Dern-Support align with the project's objectives of enhancing business operations, supporting expansion, and delivering a user-friendly experience. Continuous refinement and feedback incorporation ensure the solution's effectiveness and readiness for deployment.