Implementation of the Student Management System

Task: The goal was to create a Student Management System with the following requirements:

- **User Roles:** There are three user roles: Admin (only one), Student (multiple), and Guest (multiple).
- **Dashboards:** Each user has a customized dashboard:
    - Admin: Full access to CRUD operations and all user data.
    - Student: Access to the user list (excluding Admin) and ability to create Guest users.
    - Guest: View-only access to the user list (excluding Admin).
- **Graphical Representations:** Dashboards include role-specific graphical data views.
- **Navigation Sidebar:** Dashboards feature a sidebar for actions like profile updates, user list access, guest creation, and logout.

The project aimed to develop a comprehensive Student Management System with three distinct user roles: 'Student', 'Guest', and 'Admin'. Each user role required unique functionalities and access to a customized dashboard for seamless operation.

Implementation Details:

1. **Project Setup:**
    - Initiated a Laravel project using '**Laravel Breeze'** for authentication scaffolding.
    - Installed 'Node.js dependencies' for frontend asset compilation.
    - Integrated the '**Spatie Roles and Permissions'** package to manage user roles and permissions efficiently.
2. **Configuration:**
    - Updated 'bootstrap/app.php' with Spatie-related middleware to enable role and permission checks.
    - Modified 'app/Models/User.php' to include $fillable attributes and the HasRoles trait.
3. **Database and Seeding:**
    - Created a migration file for the users table and other required tables.
    - Developed seeders:

        - **RoleSeeder:** Predefined three roles - Admin, Student, and Guest.
        - **AdminSeeder:** Created a default Admin user for initial system access.
    - Executed migrations and seeded the database.

4. **User Registration and Validation:**
    - Enhanced create() and store() methods in ‘app/Http/Controllers/Auth/RegisteredUserController.php’ with appropriate validation rules.
    - Ensured compliance with database schema while allowing user registration.
5. **Controller Creation:**
    - Developed controllers tailored to each user role:
        - **AdminController:**
            - Full ‘CRUD’ operations for all users.
            - User list view with access to all users, including the Admin.
            - Validation through ‘AdminStoreUserRequest’ and ‘AdminUpdateUserRequest’ classes.
        - **StudentController:**
            - Access to user list excluding Admin.
            - Capability to create Guest users.
            - Validation via ‘StudentStoreGuestUserRequest’.
        - **GuestController:**
            - View-only access to the user list, excluding Admin.
6. **Frontend Development:**
    - Designed structured Blade view files for dashboards and pages based on user roles:
        - **Dashboard Views:**
            - ‘admin-dashboard.blade.php’
            - ‘student-dashboard.blade.php’
            - ‘guest-dashboard.blade.php’
        - **Page Views:**
            - ‘admin/index.blade.php’, ‘admin/create.blade.php’, ‘admin/edit.blade.php’

        - ‘student/index.blade.php’, ‘student/create.blade.php’
        - ‘guest/index.blade.php’
      - Updated default ‘welcome.blade.php’ to ‘home-page.blade.php’ for a stylish landing page.
      - Updated layouts for app, guest, and navigation views to include logos and integrate ‘Chart.js’ for graphical representations.

7. **Routing and Middleware:**
   - Defined routes for all controller methods with proper role-based middleware to ensure secure and restricted access.
8. **Graphical Representation:**
   - Implemented ‘**Chart.js’** for visual data representation on dashboards.
   - Developed methods in ‘DashboardController.php’ for data retrieval:
     - Admin: Comprehensive graphical data for all users.
     - Student: Partial graphical data.
     - Guest: Minimal graphical data.
   - Integrated JavaScript templates to render graphical views dynamically.

Technology Stack:

- The application is built on **Laravel**, leveraging the **Blade Template Engine** for frontend views.
- **JavaScript** is used for scripting and integrating dynamic graphical elements.

Key Features Delivered:

- Role-based dashboards:
  - Admin: Full access to CRUD operations and all user data.
  - Student: Access to user list (excluding Admin) and Guest user creation.
  - Guest: View-only access to user list (excluding Admin).
- Sidebar navigation for easy access to profile updates, user list, creation forms, and logout.
- Stylish and responsive UI for dashboards and the landing page.
- Secure and validated operations across all user functionalities.

Challenges and Solutions:

- Ensuring role-based access control: Addressed using Spatie’s middleware and traits.
- Creating a dynamic and responsive UI: Leveraged Laravel Blade templating and JavaScript for modular and maintainable designs.

Conclusion: The Student Management System was successfully implemented, fulfilling all the project requirements. The system is secure, scalable, and user-friendly, providing distinct experiences for Admin, Student, and Guest users. Graphical dashboards enhance data visualization, ensuring a better user experience.