Backend Developer Skill Test – Set 2 (4 Hours)

Rules & Regulations

- The duration of the test is 4 hours.
- You may use AI tools (Cursor, Claude, ChatGPT, Copilot etc.) for assistance, but ensure you understand and can explain your implementation.
- Follow Laravel best practices and the coding standards outlined here: Coding Standards Document
- Use migrations, factories, and seeders for database setup.
- All APIs must return consistent REST responses with proper HTTP status codes.
- Write test cases to validate core features and edge cases.
- Submit your solution as a compressed folder or via a file-sharing link.

Task Brief

Build an **Event Booking System** (backend).

The system should include:
- Authentication
- Event & Ticket Management
- Bookings & Payments (mocked)
- Role-based Access Control
- Middleware
- Reusable Components (Traits & Services)
- Notifications & Caching

This test evaluates skills in:
- Database Design
- Eloquent Models & Relationships
- REST API Development
- Authentication & Authorization
- Middleware, Services, Traits
- Queues & Notifications
- Caching & Optimization

Section 1: Database & Models

Time: 45 minutes

Weightage: 20%

- Models: User, Event, Ticket, Booking, Payment.
- **User**: id, name, email, password, phone, role (enum: admin, organizer, customer).
- Relationships: hasMany(Event), hasMany(Booking), hasMany(Payment).
- **Event**: id, title, description, date, location, created_by.
- Relationships: belongsTo(User), hasMany(Ticket).
- **Ticket**: id, type (VIP, Standard, etc.), price, quantity, event_id.
- Relationships: belongsTo(Event), hasMany(Booking).
- **Booking**: id, user_id, ticket_id, quantity, status (pending, confirmed, cancelled).
- Relationships: belongsTo(User), belongsTo(Ticket), hasOne(Payment).
- **Payment**: id, booking_id, amount, status (success, failed, refunded).
- Relationships: belongsTo(Booking).
- Create migrations, implement relationships, factories, and seeders.

Section 2: Authentication & Authorization

Time: 30 minutes

Weightage: 15%

- Implement Registration, Login, Logout using Sanctum.
- Protect routes with middleware.
- Role-based access control: Admin → Manage all events, tickets, bookings. Organizer → Manage their own events/tickets. Customer → Book tickets & view bookings.

Section 3: API Development

Time: 60 minutes

Weightage: 25%

- **User APIs**:
- POST /api/register
- POST /api/login
- POST /api/logout
- GET /api/me
- **Event APIs**:
- GET /api/events (pagination, search, filter by date/location)
- GET /api/events/{id} (with tickets)
- POST /api/events (organizer only)

- PUT /api/events/{id} (organizer only)
- DELETE /api/events/{id} (organizer only)
- **Ticket APIs**:
- POST /api/events/{event_id}/tickets (organizer only)
- PUT /api/tickets/{id} (organizer only)
- DELETE /api/tickets/{id} (organizer only)
- **Booking APIs**:
- POST /api/tickets/{id}/bookings (customer)
- GET /api/bookings (customer’s bookings)
- PUT /api/bookings/{id}/cancel (customer)
- **Payment APIs**:
- POST /api/bookings/{id}/payment (mock payment)
- GET /api/payments/{id}

Section 4: Middleware, Services & Traits

Time: 20 minutes

Weightage: 10%

- Custom Middleware: Prevent double booking for the same ticket.
- Service Class: PaymentService (simulate success/failure).
- Trait: CommonQueryScopes with filterByDate() and searchByTitle().

Section 5: Notifications, Queues & Caching

Time: 20 minutes

Weightage: 10%

- Notify customer when booking is confirmed.
- Use queues for sending notifications.
- Cache frequently accessed events list.

Section 6: Testing

Time: 30 minutes

Weightage: 15%

- Feature tests: Registration, Login, Event creation, Ticket booking, Payment.
- Unit test for PaymentService.
- Coverage Goal: **85%+** across controllers and services.

Section 7: Documentation & Submission

Time: 15 minutes

Weightage: 5%

- Provide Postman collection for all APIs.
- README with setup instructions.
- Include seeded data: 2 admins, 3 organizers, 10 customers, 5 events, 15 tickets, 20 bookings.