EXAM BOARD

Paper Reference: 19540/OSC/T2-ADV
Session: January 2026
Duration: 12 hours
Difficulty: Advanced

T LEVEL DIGITAL PRODUCTION, DESIGN AND DEVELOPMENT

Occupational Specialism Component - Task 2

Digital Production, Design and Development (19540)

Important Information

- This paper contains material for the completion of Task 2 of the Occupational Specialism Component
- You must complete **ALL** tasks
- Read the scenario carefully before starting
- You may use reference materials, online resources, and documentation
- All work must be your own
- You will need access to appropriate software development tools

⚠ **Technical Requirements:** This assessment requires MySQL (via XAMPP), Node.js with Express.js, and React.js. Ensure your development environment is properly configured before beginning.

Scenario: GreenLeaf Community Garden Management System

Background

GreenLeaf Community Gardens is a non-profit organisation that manages community garden allotments across multiple locations in the West Midlands

region. They currently have **147 active garden plots** spread across **6 community sites**, with over **200 registered gardeners**.

The organisation is currently managing all operations using paper records, spreadsheets, and email communications. This has become increasingly difficult as they have expanded.

Current Issues

- **Plot allocation** is tracked in multiple Excel spreadsheets that frequently become out of sync
- **Payment tracking** for annual plot rentals (£45-£120 per year depending on plot size) is done manually
- **Waiting lists** are managed via email, causing confusion about position and priority
- **Event coordination** (workshops, community days, harvest festivals) relies on email chains
- **Resource sharing** (tools, seeds, compost) has no formal tracking system

Stakeholder Requirements

The GreenLeaf management committee has identified the following key users:

User Type	Description	Estimated Count
Gardeners	Plot holders who rent garden space	200+
Site Coordinators	Volunteers managing individual sites	6
Admin Staff	Part-time administrator	1
Committee Members	Management oversight	8
Public Visitors	Potential new gardeners	Variable

System Requirements Overview

1. User Management

- Public registration with email verification
- Different access levels (Gardener, Coordinator, Admin)
- Profile management with contact preferences
- Password reset functionality

2. Plot Management

- View all plots across all sites with availability status
- Interactive site maps showing plot locations
- Plot details (size, features, annual cost, current holder)
- Historical records of plot allocations

3. Waiting List System

- Online waiting list registration
- Position tracking and estimated wait times
- Preference recording (site, plot size)
- Automatic notifications when plots become available

4. Payment Processing

- Annual rental invoicing
- Payment recording and receipts
- Outstanding payment reminders

- Payment history for each gardener

5. Event Management

- Event calendar visible to all users
- Event registration with capacity limits
- Automated reminder emails
- Attendance tracking

6. Resource Booking

- Shared tool and equipment booking
- Availability calendar
- Booking confirmation system
- Usage history and reporting

7. Photo Gallery System

- Plot photo uploads by gardeners
- Before/after transformation galleries
- Image compression and optimization
- Photo moderation by coordinators
- Seasonal showcase galleries

8. Online Payment System

- Secure online payment processing via Stripe
- Annual plot rental payments

- Event ticket purchases
- Payment receipts and invoices
- Refund processing for cancelled events
- Payment history and statements

Task 2A: Database Design

15 marks

Instructions

Design a **relational database** for the GreenLeaf Garden Management System. Your database must be implemented using **MySQL** (compatible with XAMPP).

Requirements

1. Create an Entity-Relationship Diagram (ERD) showing:

- All required entities (minimum 8 tables)
- Primary keys and foreign keys
- Relationship types (one-to-one, one-to-many, many-to-many)
- Cardinality notation

2. Produce a Data Dictionary documenting:

- Table names and descriptions
- Field names, data types, and constraints
- Primary and foreign key relationships
- Any indexes required for performance

3. Write SQL scripts to:

- Create all database tables with appropriate constraints
- Insert sample test data (minimum 10 records per main table)
- Create at least 2 stored procedures for common operations
- Create at least 1 trigger for data integrity

Minimum Required Tables

Your database **MUST** include tables for:

Users (with role differentiation)	Sites (garden locations)

Plots (individual garden plots)	Waiting List entries
Payments/Transactions	Events
Event Registrations	Resources (tools/equipment)
Resource Bookings	Photos/Gallery
Invoices	Refunds

> ⚠ **Payment Integration Note:** Store Stripe payment intents, transaction IDs, and payment status. Never store full card details - Stripe handles PCI compliance.

Mark Scheme - Task 2A

Criteria	Marks
ERD completeness and accuracy (12+ tables)	5
Appropriate normalisation (3NF)	3
Data dictionary completeness	3
SQL scripts functionality	4
Test data appropriateness	2
Payment/transaction schema design	3

Task 2B: Backend API Development — 25 marks

Instructions

Develop a **RESTful API** using **Node.js with Express.js** that provides backend services for the GreenLeaf system.

Requirements

1. Project Setup

- Initialize a Node.js project with appropriate package.json
- Use Express.js as the web framework
- Implement proper project structure (routes, controllers, models, middleware)
- Configure environment variables for database connection

2. Database Integration

- Connect to your MySQL database
- Implement a connection pool for efficient database access
- Use parameterized queries to prevent SQL injection

3. API Endpoints

Authentication Routes (`/api/auth`)

Method	Endpoint	Description
POST	`/register`	Register new user
POST	`/login`	User login with session/JWT
POST	`/logout`	End user session
POST	`/forgot-password`	Request password reset
POST	`/reset-password`	Reset password with token

User Routes (`/api/users`)

Method	Endpoint	Description
GET	`/profile`	Get current user profile
PUT	`/profile`	Update user profile
GET	`/:id`	Get user by ID (admin only)
GET	`/`	List all users (admin only)

Plot Routes (`/api/plots`)

Method	Endpoint	Description
GET	`/`	List all plots with filters
GET	`/:id`	Get plot details
GET	`/site/:siteId`	Get plots by site
POST	`/`	Create new plot (admin)
PUT	`/:id`	Update plot (admin)
PUT	`/:id/assign`	Assign plot to user (admin)

Waiting List Routes (`/api/waitlist`)

Method	Endpoint	Description
POST	`/join`	Join the waiting list
GET	`/position`	Get current user's position
GET	`/`	List all entries (admin)
DELETE	`/:id`	Remove from waiting list

Event Routes (`/api/events`)

Method	Endpoint	Description
GET	`/`	List all upcoming events
GET	`/:id`	Get event details
POST	`/`	Create event (coordinator+)
PUT	`/:id`	Update event (coordinator+)
POST	`/:id/register`	Register for event
DELETE	`/:id/register`	Cancel registration

Resource Routes (`/api/resources`)

Method	Endpoint	Description
GET	`/`	List all resources
GET	`/:id/availability`	Check resource availability
POST	`/bookings`	Book a resource
GET	`/bookings/my`	Get user's bookings
DELETE	`/bookings/:id`	Cancel booking

Photo/Gallery Routes (`/api/photos`)

Method	Endpoint	Description
GET	`/`	List all approved photos
GET	`/plot/:plotId`	Get photos for a specific plot
POST	`/upload`	Upload photo (multipart/form-data)
DELETE	`/:id`	Delete own photo
PUT	`/:id/approve`	Approve photo (coordinator+)
PUT	`/:id/reject`	Reject photo (coordinator+)

Method	Endpoint	Description
GET	`/pending`	List pending photos (coordinator+)

Payment Routes (`/api/payments`) - *Stripe Integration*

Method	Endpoint	Description
POST	`/create-intent`	Create Stripe payment intent
POST	`/confirm`	Confirm payment completion
GET	`/history`	Get user's payment history
GET	`/invoice/:id`	Get invoice details
GET	`/invoice/:id/pdf`	Download invoice as PDF
POST	`/refund/:id`	Request refund (admin)
POST	`/webhook`	Stripe webhook handler

File Upload Requirements

- Use `multer` middleware for handling multipart/form-data
- Validate file types (JPEG, PNG, WebP only)
- Limit file size to 5MB maximum
- Generate unique filenames to prevent conflicts
- Store files in `/uploads/photos/` directory
- Create thumbnail versions (300x300) for gallery views

Stripe Integration Requirements

- Use Stripe API in **test mode** (test API keys)
- Implement Payment Intents API for SCA compliance

- Handle webhook events for payment confirmation
- Support payment for: plot rentals, event tickets
- Generate and store transaction references
- Implement proper error handling for failed payments

4. Middleware Implementation

- Authentication middleware (verify logged-in users)
- Authorization middleware (role-based access control)
- Error handling middleware
- Request logging middleware
- Input validation middleware

5. Security Requirements

- Password hashing using bcrypt
- Session management OR JWT tokens
- CORS configuration
- Rate limiting on authentication routes
- Input sanitization

Mark Scheme - Task 2B

Criteria	Marks
Project structure and organization	3
Database connection and queries	4
Authentication implementation	5
CRUD operations completeness	5
Middleware implementation	4
Error handling and validation	2

Criteria	Marks
Security best practices	2
File upload implementation (multer)	4
Stripe payment integration	6

Task 2C: Frontend Development — 25 marks

Instructions

Develop a **Single Page Application (SPA)** using **React.js** that provides the user interface for the GreenLeaf system.

Requirements

1. Project Setup

- Initialize a React project (Vite or Create React App)
- Implement component-based architecture
- Set up React Router for navigation
- Configure environment variables for API URL

2. Core Pages/Views

Page	Route	Description	Access
Home	`/`	Landing page with system info	Public
Login	`/login`	User authentication	Public
Register	`/register`	New user registration	Public
Dashboard	`/dashboard`	User overview	Authenticated
Profile	`/profile`	View/edit profile	Authenticated
Plots	`/plots`	Browse available plots	Public
Plot Details	`/plots/:id`	Individual plot info	Public
Waiting List	`/waitlist`	Join/view position	Authenticated
Events	`/events`	Event calendar/list	Public

Page	Route	Description	Access
Event Details	`/events/:id`	Event info & registration	Public
Resources	`/resources`	Resource booking	Authenticated
Gallery	`/gallery`	Photo gallery showcase	Public
My Photos	`/photos`	Upload and manage photos	Authenticated
Payments	`/payments`	Make payments & view history	Authenticated
Checkout	`/checkout/:type/:id`	Stripe checkout flow	Authenticated
Payment Success	`/payment/success`	Payment confirmation	Authenticated
Payment Failed	`/payment/failed`	Payment failure handling	Authenticated
Admin Panel	`/admin`	User/plot management	Admin only
Photo Moderation	`/admin/photos`	Approve/reject photos	Coordinator+

3. Component Requirements

Create reusable components including:

- Navigation bar with role-based menu items
- Authentication forms (login, register, password reset)
- Plot card/grid components
- Event calendar or list view
- Resource booking calendar
- Data tables with sorting/filtering
- Modal dialogs for confirmations
- Loading spinners and error messages
- Flash/toast notifications

Photo Gallery Components

- Image upload component with drag-and-drop support

- Image preview before upload
- Photo gallery grid with lightbox view
- Photo card with metadata (plot, date, gardener)
- Image cropping/rotation tool (optional)
- Upload progress indicator
- Photo moderation queue (coordinator view)

Payment Components (Stripe Integration)

- Payment summary card showing amount due
- Stripe Elements integration for secure card input
- Payment processing loading state
- Payment success/failure screens
- Invoice display and download button
- Payment history table
- Refund request modal (if applicable)

> ⚠ **Stripe Frontend Integration:** Use `@stripe/stripe-js` and `@stripe/react-stripe-js` packages. Never handle raw card numbers - use Stripe Elements for PCI compliance.

4. State Management

- Implement Context API for authentication state and flash messages
- Proper state handling for forms
- API response caching where appropriate

5. UI/UX Requirements

- Responsive design (mobile, tablet, desktop)
- Consistent styling throughout
- Form validation with user feedback
- Accessible navigation (keyboard, screen readers)

- Professional, clean appearance

Mark Scheme - Task 2C

Criteria	Marks
Project structure and components	4
Routing implementation	3
Authentication flow	4
Data display and management	5
Form handling and validation	3
State management	3
UI/UX and responsiveness	3
Photo upload and gallery implementation	5
Stripe Elements checkout integration	5

Task 2D: Integration and Testing

15 marks

Instructions

Integrate all components of your system and conduct comprehensive testing.

Requirements

1. System Integration

- Ensure frontend successfully communicates with backend
- Verify all CRUD operations work end-to-end
- Test authentication flow from registration to protected routes
- Confirm role-based access works correctly

2. Testing Documentation

Create a **Test Plan** with minimum **30 test cases** covering:

- User authentication (5+ tests)
- Plot management (4+ tests)
- Waiting list functionality (3+ tests)
- Event operations (4+ tests)
- Resource booking (4+ tests)
- Photo upload and gallery (5+ tests)
- Payment processing with Stripe (5+ tests)

> #### Payment Testing with Stripe
>
> Use Stripe's test card numbers for payment testing:
>
> - `4242 4242 4242 4242` - Successful payment
> - `4000 0000 0000 0002` - Card declined

> - `4000 0000 0000 9995` - Insufficient funds
>
> Document test results for each scenario.

Test ID	Feature	Test Case	Expected Result	Actual Result	Status
T001	Registration	Valid user registration	User created, redirected to login		
T002	Registration	Duplicate email	Error message displayed		
...	...	...	...		

3. Bug Tracking

Document any bugs found during testing:

Bug ID	Description	Severity	Steps to Reproduce	Status
B001				

4. Screenshots/Evidence

- Capture screenshots of key functionality working
- Include evidence of both successful and error states
- Show different user role interfaces

Mark Scheme - Task 2D

Criteria	Marks
Successful integration	5
Test plan completeness	4
Test execution and evidence	4

Criteria	Marks
Bug identification and documentation	2

Task 2E: Documentation and Evaluation — 10 marks

Instructions

Produce professional documentation and critically evaluate your solution.

Requirements

1. Technical Documentation

- System architecture diagram
- API documentation (endpoints, parameters, responses)
- Database schema documentation
- Installation/setup instructions

2. User Guide

- Getting started guide for new users
- Feature overview with screenshots
- FAQ section

3. Critical Evaluation (500-750 words)

Address the following:

- How well does the solution meet the stakeholder requirements?
- What technical challenges were encountered and how were they solved?
- What security measures were implemented and why?
- What improvements would you make given more time?
- How does the solution compare to alternative approaches?
- What did you learn from this project?

Mark Scheme - Task 2E

Criteria	Marks
Technical documentation quality	3
User guide clarity and completeness	3
Evaluation depth and insight	4

TOTAL MARKS: 120

Task	Marks
Task 2A: Database Design	20
Task 2B: Backend API Development (inc. File Upload & Stripe)	35
Task 2C: Frontend Development (inc. Gallery & Payments)	35
Task 2D: Integration and Testing	18
Task 2E: Documentation and Evaluation	12
Total	**120**

Grade Boundaries (Indicative)

Grade	Mark Range
Distinction*	108-120 (90%+)
Distinction	96-107 (80-89%)
Merit	72-95 (60-79%)
Pass	48-71 (40-59%)
Near Pass	36-47 (30-39%)

Grade	Mark Range
Unclassified	0-35 (<30%)

Appendix A: Technical Environment

Recommended Software Stack

Component	Technology
Database	MySQL 8.0+ (via XAMPP)
Backend	Node.js 18+, Express.js 4+
Frontend	React 18+, React Router 6+
File Upload	Multer, Sharp (image processing)
Payments	Stripe API (stripe, @stripe/stripe-js)
Development	VS Code, Git

Required NPM Packages

Backend

```
npm install express mysql2 bcrypt express-session cors dotenv
npm install multer sharp uuid
npm install stripe
```

Frontend

```
npm install react-router-dom axios
npm install @stripe/stripe-js @stripe/react-stripe-js
```

Stripe Setup

1. Create a free Stripe account at `https://stripe.com`
2. Access the Dashboard and find your **Test API keys**
3. Copy the `Publishable key` (starts with `pk_test_`) for frontend
4. Copy the `Secret key` (starts with `sk_test_`) for backend
5. Set up a webhook endpoint for payment confirmations

6. **Never commit API keys to version control!**

XAMPP Setup Notes

1. Install XAMPP with MySQL/MariaDB
2. Start Apache and MySQL services
3. Access phpMyAdmin at `http://localhost/phpmyadmin`
4. Create database named `greenleaf_gardens`
5. Default connection: `localhost:3306`, user `root`, no password

Appendix B: Sample Data

Sites

Site ID	Name	Address	Total Plots
1	Riverside Gardens	45 River Lane, Birmingham B1 2CD	32
2	Hilltop Allotments	12 Summit Road, Coventry CV1 3EF	28
3	Meadow View	78 Green Street, Wolverhampton WV2 4GH	24

Plot Sizes and Pricing

Size	Dimensions	Annual Cost
Small	5m x 10m	£45
Medium	10m x 10m	£75
Large	10m x 20m	£120

Sample Events

Event	Date	Capacity
Spring Planting Workshop	March 15	20

Event	Date	Capacity
Composting Masterclass	April 22	15
Summer Garden Party	July 10	100
Harvest Festival	September 18	150

Appendix C: Wireframe Suggestion

Dashboard Layout

End of Assessment Paper

Paper Reference: 19540/OSC/T2 | Session: January 2026