🎾 Development and Integration Phase

Vertical Slicing Approach for Padel Booking Platform

Development Methodology	Total Duration	Team Size
Vertical Slicing	**14 Weeks**	**4 Developers**

1. Development Methodology: Vertical Slicing

What is Vertical Slicing?

Vertical slicing is an agile development approach where features are built end-to-end through all layers of the architecture in each iteration. Instead of building horizontal layers (all backend, then all frontend), we deliver complete, working features that span from UI to database.

Benefits for Our Microservices Architecture:

- ✅ **Early Value Delivery:** Each slice is a complete, deployable feature
- ✅ **Reduced Risk:** Problems discovered early through full-stack integration
- ✅ **Better Feedback:** Stakeholders can see and test real features quickly
- ✅ **Parallel Development:** Teams can work on different slices simultaneously
- ✅ **Microservices Alignment:** Each slice can span multiple services naturally

Our Slicing Strategy:

1. **Thin Slices First:** Start with minimal viable features
2. **Cross-Service Slices:** Each slice integrates relevant microservices
3. **User Journey Focus:** Slices follow complete user workflows
4. **Progressive Enhancement:** Each iteration adds depth to existing slices

2. Development Phases Overview

Phase 0: Foundation & Infrastructure

Week 1-2

Set up core infrastructure, CI/CD pipelines, and development environment

Phase 1: Core Authentication Slice

Week 3-4

Complete user registration, login, and session management across all services

Phase 2: Basic Booking Slice

Week 5-7

End-to-end booking flow with court availability and basic confirmations

Phase 3: Payment Integration Slice

Week 8-9

Complete payment processing with Stripe and local gateways

Phase 4: Notification Slice

Week 10-11

Multi-channel notifications including WhatsApp integration

Phase 5: Admin Dashboard Slice

Week 12-13

Club management interface with analytics and controls

Phase 6: Integration & Polish

Week 14

Final integration, testing, and production deployment preparation

3. Phase 0: Foundation & Infrastructure (Week 1-2)

Objectives

Establish the complete development infrastructure and CI/CD pipeline before any feature development. This phase ensures all developers can work efficiently with proper tooling and automation.

Infrastructure Setup Critical

Tasks:

- [ ] Set up Kubernetes cluster (DigitalOcean/AWS EKS)
- [ ] Configure namespaces: dev, staging, production
- [ ] Install Ingress Controller (NGINX)
- [ ] Set up cert-manager for SSL certificates
- [ ] Configure DNS and domain routing

- Set up managed PostgreSQL instances
- Deploy Redis cluster for caching
- Configure S3 buckets for media storage

```
# Kubernetes namespace configuration apiVersion: v1 kind: Namespace metadata:
name: padel-dev --- apiVersion: v1 kind: Namespace metadata: name: padel-
staging --- apiVersion: v1 kind: Namespace metadata: name: padel-production
```

CI/CD Pipeline Critical

GitLab CI Pipeline Structure:

```
stages: - test - build - deploy-dev - deploy-staging - deploy-production
variables: DOCKER_REGISTRY: registry.gitlab.com/padel-booking
K8S_NAMESPACE_DEV: padel-dev K8S_NAMESPACE_STAGING: padel-staging
K8S_NAMESPACE_PROD: padel-production # Template for microservice builds
.service-build: stage: build image: docker:latest services: - docker:dind
script: - docker build -t $DOCKER_REGISTRY/$SERVICE_NAME:$CI_COMMIT_SHA . -
docker push $DOCKER_REGISTRY/$SERVICE_NAME:$CI_COMMIT_SHA - docker tag
$DOCKER_REGISTRY/$SERVICE_NAME:$CI_COMMIT_SHA
$DOCKER_REGISTRY/$SERVICE_NAME:latest - docker push
$DOCKER_REGISTRY/$SERVICE_NAME:latest # Automated testing test: stage: test
```

```
script: - npm install - npm run test:unit - npm run test:integration coverage:
'/Coverage: \d+\.\d+%/'
```

Deliverables:

✅ Phase 0 Deliverables

- Fully configured Kubernetes cluster with 3 environments
- Automated CI/CD pipeline with GitLab CI
- Docker registry for container images
- Database instances provisioned and secured
- Monitoring stack (Prometheus + Grafana) deployed
- Development environment documentation

🛠️ Development Environment Critical

Local Development Setup:

```
# docker-compose.dev.yml for local development version: '3.8' services: auth-
service: build: ./services/auth ports: - "3001:3001" environment: -
NODE_ENV=development -
DATABASE_URL=postgresql://user:pass@postgres:5432/auth_db -
REDIS_URL=redis://redis:6379 volumes: - ./services/auth:/app -
/app/node_modules depends_on: - postgres - redis user-service: build:
```

```
./services/user ports: - "3002:3002" environment: - NODE_ENV=development -
DATABASE_URL=postgresql://user:pass@postgres:5432/user_db volumes: -
./services/user:/app - /app/node_modules depends_on: - postgres booking-
service: build: ./services/booking ports: - "3003:3003" environment: -
NODE_ENV=development -
DATABASE_URL=postgresql://user:pass@postgres:5432/booking_db -
REDIS_URL=redis://redis:6379 volumes: - ./services/booking:/app -
/app/node_modules depends_on: - postgres - redis notification-service: build:
./services/notification ports: - "3004:3004" environment: -
NODE_ENV=development - REDIS_URL=redis://redis:6379 volumes: -
./services/notification:/app - /app/node_modules depends_on: - redis postgres:
image: postgres:15-alpine environment: - POSTGRES_USER=user -
POSTGRES_PASSWORD=pass - POSTGRES_MULTIPLE_DATABASES=auth_db,user_db,booking_db
ports: - "5432:5432" volumes: - postgres_data:/var/lib/postgresql/data redis:
image: redis:7-alpine ports: - "6379:6379" rabbitmq: image: rabbitmq:3-
management ports: - "5672:5672" - "15672:15672" environment: -
RABBITMQ_DEFAULT_USER=admin - RABBITMQ_DEFAULT_PASS=admin volumes:
postgres_data:
```

4. Phase 1: Core Authentication Slice (Week 3-4)

Vertical Slice: Complete Authentication Flow

Build end-to-end authentication spanning all layers: Frontend → API Gateway → Auth Service → Database

User Registration Flow Sprint 1

Frontend Tasks:

- Create registration form with validation
- Implement phone number verification UI
- Add password strength indicator
- Create success/error feedback components

Backend Tasks:

- Auth Service: Registration endpoint
- User Service: Create user profile endpoint
- SMS integration for OTP verification
- Event: user.registered published to message bus

Database Tasks:

- Create users table in User Service DB
- Create sessions table in Auth Service DB
- Set up Redis for OTP storage

📋 Acceptance Criteria

- User can register with email/phone
- Phone verification via OTP works
- Duplicate email/phone rejected
- Welcome email sent upon registration
- User profile created in User Service

🔐 Login & Session Management Sprint 1

Implementation Details:

```
// Auth Service - Login Endpoint @Post('/login') async login(@Body() dto: LoginDto) { const user = await this.authService.validateUser(dto); if (!user) { throw new UnauthorizedException('Invalid credentials'); } const tokens = await this.authService.generateTokens(user); // Store refresh token in Redis await this.redis.set( `refresh_token:${user.id}`, tokens.refreshToken, 'EX', 7 * 24 * 60 * 60 // 7 days ); // Publish login event await this.eventBus.publish('user.logged_in', { userId: user.id, timestamp: new Date(), ip: req.ip }); return { accessToken: tokens.accessToken, refreshToken: tokens.refreshToken, user: { id: user.id, email: user.email, name: user.name } }; }
```

Security Features:

- JWT with RS256 signing
- Refresh token rotation
- Rate limiting on login attempts
- Account lockout after failed attempts
- Session management in Redis

✅ Phase 1 Deliverables

- Complete registration and login flow
- JWT authentication working across all services
- Phone/email verification implemented
- Password reset functionality
- Session management with Redis
- Auth middleware for all services

5. Phase 2: Basic Booking Slice (Week 5-7)

Vertical Slice: Court Booking Journey

Complete booking flow from court selection to confirmation, integrating multiple services

Court Availability & Selection Sprint 2

Frontend Components:

- Calendar view for date selection
- Time slot grid showing availability
- Court selection interface
- Real-time availability updates via WebSocket
- Mobile-responsive booking interface

Booking Service Implementation:

```
// Booking Service - Check Availability @Get('/courts/:clubId/availability')
async getAvailability( @Param('clubId') clubId: string, @Query('date') date:
string ) { // Check cache first const cacheKey =
`availability:${clubId}:${date}`; const cached = await
this.redis.get(cacheKey); if (cached) { return JSON.parse(cached); } // Query
database for court availability const courts = await
this.courtRepository.find({ clubId }); const bookings = await
this.bookingRepository.find({ where: { courtId: In(courts.map(c => c.id)),
date: date, status: Not('cancelled') } }); // Calculate available slots const
availability = this.calculateAvailability(courts, bookings); // Cache for 5
minutes await this.redis.setex(cacheKey, 300, JSON.stringify(availability)); //
Set up real-time subscription await
this.pubsub.publish(`availability:${clubId}`, availability); return
availability; } // Real-time availability updates @WebSocketGateway() export
class AvailabilityGateway { @SubscribeMessage('subscribe:availability') async
handleSubscription(client: Socket, payload: { clubId: string, date: string }) {
const room = `availability:${payload.clubId}:${payload.date}`;
client.join(room); // Send current availability const availability = await
this.bookingService.getAvailability( payload.clubId, payload.date );
client.emit('availability:update', availability); } }
```

🎾 Booking Creation Flow Sprint 3

Booking Flow Steps:

> User selects court and time slot

System validates availability (with lock)

Booking created with 'pending' status

Payment initiated (Phase 3)

Booking confirmed upon payment

Confirmation sent to user

Event Flow:

```
// Booking Service - Create Booking async createBooking(dto: CreateBookingDto)
{ // Start database transaction const booking = await this.db.transaction(async
(manager) => { // Lock the time slot to prevent double booking const slot =
await manager.findOne(CourtAvailability, { where: { courtId: dto.courtId, date:
dto.date, timeSlot: dto.timeSlot, isAvailable: true }, lock: { mode:
'pessimistic_write' } }); if (!slot) { throw new ConflictException('Slot no
longer available'); } // Create booking const booking = await
manager.save(Booking, { userId: dto.userId, courtId: dto.courtId, startTime:
dto.startTime, endTime: dto.endTime, status: 'pending', amount: dto.amount });
// Mark slot as unavailable slot.isAvailable = false; await manager.save(slot);
return booking; }); // Publish booking created event await
this.eventBus.publish('booking.created', { bookingId: booking.id, userId:
booking.userId, courtId: booking.courtId, amount: booking.amount, startTime:
booking.startTime }); // Set timeout for payment (15 minutes) await
```

```
this.queue.add('booking.payment.timeout', { bookingId: booking.id }, { delay:
15 * 60 * 1000 }); return booking; }
```

📱 Booking Management Sprint 3

User Features:

- View upcoming bookings
- Cancel booking (with refund policy)
- Modify booking time
- Booking history
- Add to calendar (Google/Apple)

✅ Phase 2 Deliverables

- Complete booking flow from selection to confirmation
- Real-time availability updates
- Booking management interface
- Court and club data management

- Prevent double bookings with locking
- Basic cancellation flow

6. Phase 3: Payment Integration Slice (Week 8-9)

Vertical Slice: Complete Payment Flow

Integrate payment processing with Stripe and local payment gateways

💳 Payment Processing Sprint 4

Payment Gateway Integration:

```
// Payment Module in Booking Service (MVP approach) export class PaymentService
{ private stripe: Stripe; private easyPaisa: EasyPaisaClient; async
processPayment(booking: Booking, method: PaymentMethod) { let result; switch
(method.type) { case 'card': result = await this.processStripePayment(booking,
method); break; case 'easypaisa': result = await
this.processEasyPaisaPayment(booking, method); break; case 'jazzcash': result =
await this.processJazzCashPayment(booking, method); break; default: throw new
BadRequestException('Invalid payment method'); } // Save payment record const
payment = await this.paymentRepository.save({ bookingId: booking.id, userId:
booking.userId, amount: booking.amount, method: method.type, status:
result.status, gatewayTransactionId: result.transactionId, metadata:
result.metadata }); // Update booking status if payment successful if
(result.status === 'completed') { booking.status = 'confirmed';
booking.paymentId = payment.id; await this.bookingRepository.save(booking); //
```

```
Publish payment completed event await
this.eventBus.publish('payment.completed', { paymentId: payment.id, bookingId:
booking.id, userId: booking.userId, amount: payment.amount }); } return
payment; } private async processStripePayment(booking: Booking, method: any) {
const paymentIntent = await this.stripe.paymentIntents.create({ amount:
booking.amount * 100, // Convert to cents currency: 'pkr', customer:
method.customerId, payment_method: method.paymentMethodId, confirm: true,
metadata: { bookingId: booking.id, userId: booking.userId } }); return {
status: paymentIntent.status === 'succeeded' ? 'completed' : 'failed',
transactionId: paymentIntent.id, metadata: paymentIntent }; } }
```

Webhook Handlers:

- Stripe webhook for payment confirmations
- EasyPaisa callback handler
- JazzCash IPN listener
- Webhook signature verification
- Idempotency handling

💰 Refund Management Sprint 4

Refund Policy Implementation:

24-hour cancellation: 100% refund

12-hour cancellation: 50% refund

Less than 12 hours: No refund

Automatic refund processing

Refund status tracking

✅ Phase 3 Deliverables

- Stripe integration for card payments
- EasyPaisa integration
- JazzCash integration
- Payment confirmation flow
- Refund processing
- Payment history and receipts

7. Phase 4: Notification Slice (Week 10-11)

Vertical Slice: Multi-Channel Notifications

Implement complete notification system including WhatsApp, SMS, and Email

Notification Service Implementation Sprint 5

Notification Types:

Event	Email	SMS	WhatsApp	Push
Booking Confirmation	✅	✅	✅	✅
Payment Receipt	✅	-	✅	-
Booking Reminder	-	✅	✅	✅
Cancellation Notice	✅	✅	✅	✅
Welcome Message	✅	-	✅	-

WhatsApp Integration (Notifications Only):

```
// Notification Service - WhatsApp Handler export class
WhatsAppNotificationHandler { private client: WhatsAppBusinessAPI; async
sendBookingConfirmation(booking: BookingConfirmedEvent) { const user = await
this.userService.getUser(booking.userId); const message = { to:
user.whatsappNumber || user.phone, type: 'template', template: { name:
'booking_confirmation', language: { code: 'en' }, components: [ { type: 'body',
parameters: [ { type: 'text', text: user.name }, { type: 'text', text:
booking.courtName }, { type: 'text', text: formatDate(booking.startTime) }, {
type: 'text', text: formatTime(booking.startTime) }, { type: 'text', text:
booking.bookingId } ] } ] } }; try { const response = await
this.client.messages.send(message); // Log notification sent await
this.notificationLog.create({ userId: user.id, type: 'whatsapp', template:
'booking_confirmation', status: 'sent', messageId: response.messageId,
timestamp: new Date() }); } catch (error) { // Fallback to SMS await
this.smsHandler.send(user.phone, `Booking confirmed! Court:
${booking.courtName}, ` + `Date: ${formatDate(booking.startTime)}, ` + `Time:
${formatTime(booking.startTime)}` ); } } }
```

🔔 Notification Queue Management Sprint 5

Queue Implementation with BullMQ:

> Priority queues for urgent notifications

Retry logic with exponential backoff

Dead letter queue for failed messages

Rate limiting per channel

Batch processing for efficiency

✅ Phase 4 Deliverables

- Multi-channel notification system
- WhatsApp notifications (templates)
- SMS integration via Twilio
- Email templates with SendGrid
- Notification preferences management
- Delivery tracking and logs

8. Phase 5: Admin Dashboard Slice (Week 12-13)

Vertical Slice: Club Management Interface

Complete admin dashboard for club management and analytics

📊 Admin Dashboard Features

Sprint 6

Dashboard Components:

- Real-time booking overview
- Court availability management
- Revenue analytics and reports
- Customer management interface
- Pricing configuration
- Court maintenance scheduling

Analytics Implementation:

```
// Admin Dashboard - Analytics API @Get('/analytics/dashboard')
@UseGuards(AdminGuard) async getDashboardAnalytics(@Query() query:
AnalyticsQueryDto) { const { startDate, endDate, clubId } = query; // Parallel
queries for performance const [ bookingStats, revenueData, utilizationRate,
customerMetrics, popularTimeSlots ] = await Promise.all([
this.getBookingStatistics(clubId, startDate, endDate),
this.getRevenueAnalytics(clubId, startDate, endDate),
this.getCourtUtilization(clubId, startDate, endDate),
this.getCustomerMetrics(clubId, startDate, endDate),
this.getPopularTimeSlots(clubId) ]); return { period: { startDate, endDate },
bookings: bookingStats, revenue: revenueData, utilization: utilizationRate,
customers: customerMetrics, insights: { popularSlots: popularTimeSlots,
recommendations: this.generateRecommendations(utilizationRate) } }; } private
async getCourtUtilization(clubId: string, startDate: Date, endDate: Date) {
const totalSlots = await this.calculateTotalSlots(clubId, startDate, endDate);
const bookedSlots = await this.bookingRepository.count({ where: { clubId,
startTime: Between(startDate, endDate), status: In(['confirmed', 'completed'])
} }); return { totalSlots, bookedSlots, utilizationRate: (bookedSlots /
totalSlots) * 100, trend: await this.calculateTrend(clubId, startDate, endDate)
}; }
```

⚙ Club Configuration Management Sprint 6

Configuration Features:

- Operating hours management
- Peak/off-peak pricing
- Court-specific settings
- Booking rules (max duration, advance booking)
- Holiday and special event configuration

✅ Phase 5 Deliverables

- Complete admin dashboard
- Real-time analytics and reporting
- Court management interface
- Customer management system
- Revenue tracking and reports
- Configuration management

9. Phase 6: Integration & Polish (Week 14)

Final Integration and Production Readiness

Complete system integration, performance optimization, and production deployment preparation

🔧 System Integration Testing

Final Sprint

Integration Test Scenarios:

- Complete booking flow with payment
- Cancellation and refund process
- Multi-user concurrent bookings
- Notification delivery across all channels
- Admin operations impact on user experience
- Service failure and recovery scenarios

Performance Testing:

```
# K6 Load Testing Script import http from 'k6/http'; import { check, sleep } from 'k6'; export let options = { stages: [ { duration: '2m', target: 100 }, // Ramp up to 100 users { duration: '5m', target: 100 }, // Stay at 100 users { duration: '2m', target: 200 }, // Ramp up to 200 users { duration: '5m', target: 200 }, // Stay at 200 users { duration: '2m', target: 0 }, // Ramp down to 0 users ], thresholds: { http_req_duration: ['p(95)<2000'], // 95% of requests under 2s http_req_failed: ['rate<0.1'], // Error rate under 10% }, }; export default function() { // Test booking flow let response = http.get('https://api.padel-booking.pk/courts/availability'); check(response, { 'status is 200': (r) => r.status === 200, 'response time < 2s': (r) => r.timings.duration < 2000, }); sleep(1); }
```

🚀 Production Deployment Preparation

Final Sprint

Deployment Checklist:

- SSL certificates configured
- Environment variables secured in Vault
- Database backups configured
- Monitoring alerts set up

- Log aggregation verified
- Rate limiting configured
- CDN cache rules set
- Health checks implemented
- Rollback procedures documented

📚 Documentation & Training

Final Sprint

Documentation Deliverables:

- API documentation (OpenAPI/Swagger)
- Admin user guide
- Player onboarding guide
- Troubleshooting runbook
- Architecture diagrams

Database schema documentation

✅ Phase 6 Deliverables

- Production-ready application
- Complete test coverage (>80%)
- Performance benchmarks met
- Security audit passed
- Documentation complete
- Deployment procedures verified
- Training materials prepared

10. Sprint Planning & Timeline

Sprint	Week	Primary Focus	Key Deliverables	Team Allocation
Sprint 0	1-2	Infrastructure Setup	K8s cluster, CI/CD, Dev environment	Full team (4 devs)
Sprint 1	3-4	Authentication	Registration, Login, JWT	2 Backend, 1 Frontend, 1 DevOps
Sprint 2	5-6	Booking Core	Availability, Court selection	2 Backend, 2 Frontend
Sprint 3	7	Booking Management	Booking CRUD, Cancellation	1 Backend, 2 Frontend, 1 Testing
Sprint 4	8-9	Payments	Stripe, EasyPaisa, Refunds	2 Backend, 1 Frontend, 1 Testing
Sprint 5	10-11	Notifications	Email, SMS, WhatsApp	2 Backend, 1 Frontend, 1 DevOps
Sprint 6	12-13	Admin Dashboard	Analytics, Management UI	1 Backend, 2 Frontend, 1 Testing

Sprint	Week	Primary Focus	Key Deliverables	Team Allocation
Sprint 7	14	Integration & Polish	Testing, Optimization, Deployment	Full team (4 devs)

11. Success Metrics & KPIs

📈 Development KPIs

- **Sprint Velocity:** Complete 40-50 story points per sprint
- **Code Coverage:** Maintain >80% test coverage
- **Build Success Rate:** >95% CI/CD pipeline success
- **Bug Discovery Rate:** <5 critical bugs per sprint
- **Performance:** All API endpoints <500ms response time
- **Availability:** 99.9% uptime in staging environment

🎯 Business Metrics (MVP Launch)

- **Feature Completion:** 100% of MVP features delivered
- **User Acceptance:** >90% satisfaction in UAT
- **Performance:** Support 100 concurrent users
- **Mobile Responsiveness:** Works on 95% of devices
- **Payment Success:** >98% transaction success rate

- **Notification Delivery:** >99% delivery rate

12. Risk Management

⚠️ Technical Risks & Mitigation

Risk	Impact	Probability	Mitigation
Microservices complexity	High	Medium	Start with 4 services only, add gradually
Payment gateway delays	High	Low	Early integration, multiple gateway options
WhatsApp API approval	Medium	Medium	Apply early, SMS fallback ready
Performance issues	High	Low	Load testing each sprint, caching strategy
Team skill gaps	Medium	Medium	Training budget, pair programming

13. Definition of Done

✅ Feature Complete When:

- Code reviewed and approved by at least one team member
- Unit tests written and passing (>80% coverage)
- Integration tests passing
- API documentation updated
- No critical or high-priority bugs
- Performance benchmarks met
- Security scan passed
- Deployed to staging environment
- Product owner acceptance received

🎉 MVP Complete Checklist

- ✅ All 4 microservices deployed and stable
- ✅ User can register, login, and manage profile
- ✅ Complete booking flow from selection to confirmation
- ✅ Payment processing with Stripe and local gateways
- ✅ Multi-channel notifications working
- ✅ Admin dashboard with analytics
- ✅ 99.9% uptime achieved
- ✅ <2 second response time for all operations
- ✅ Security audit passed

- ✅ Production deployment ready