# Reservation Microservice — 25 Commit Plan (Node.js)

This document provides a detailed, incremental 25-commit roadmap for building the **Reservation Microservice** using Node.js, Express, PostgreSQL, and modern tooling.

Each commit is designed to be atomic, testable, and builds upon the previous one. Follow them in order for a smooth development experience.

---

## Technology Stack

- **Runtime:** Node.js (v18+ LTS)
- **Framework:** Express.js
- **Database:** PostgreSQL (using `pg` or `pg-promise`)
- **Auth:** JWT (jsonwebtoken) + bcrypt
- **Validation:** Joi or express-validator
- **QR Code:** qrcode library
- **Testing:** Jest + Supertest
- **Process Manager:** PM2 (optional)
- **Migrations:** node-pg-migrate or Knex.js
- **Logging:** Winston or Pino
- **Environment:** dotenv

---

## 25-Commit Breakdown

### Phase 1: Project Setup & Infrastructure (Commits 1-5)

#### Commit 1: Initialize Node.js project and dependencies

- **What to do:**
  - Run `npm init -y` to create `package.json`
  - Install core dependencies: `express`, `pg`, `dotenv`, `cors`, `helmet`
  - Install dev dependencies: `nodemon`, `jest`, `supertest`, `eslint`, `prettier`
  - Create `.gitignore` (node_modules, .env, logs, coverage)
  - Create `README.md` with project overview
- **Files:** `package.json`, `.gitignore`, `README.md`
- **Commit message:** `chore: initialize Node.js project with core dependencies`

#### Commit 2: Setup project folder structure

- **What to do:**
  - Create folder structure:

```
backend/reservation-service/
├── src/
│   ├── config/
│   ├── controllers/
│   ├── middlewares/
│   ├── models/
│   ├── routes/
│   ├── services/
│   ├── utils/
│   └── app.js
├── tests/
├── migrations/
├── .env.example
└── server.js
```

    - Create placeholder files with basic exports
- **Files:** folder structure, empty placeholder files
- **Commit message:** `chore: setup project folder structure`

**Commit 3: Configure database connection and environment**

- **What to do:**
    - Create `src/config/database.js` with PostgreSQL connection pool
    - Create `.env.example` with DB credentials, PORT, JWT_SECRET
    - Create `.env` (gitignored) with actual values
    - Test database connection with a simple query
- **Files:** `src/config/database.js`, `.env.example`, `.env`
- **Commit message:** `feat: configure PostgreSQL database connection`

**Commit 4: Setup Express app with middleware**

- **What to do:**
    - Create `src/app.js` with Express instance
    - Add middleware: `express.json()`, `cors()`, `helmet()`
    - Add basic health check route: `GET /health`
    - Create `server.js` to start the server
    - Add npm scripts: `start`, `dev` (nodemon)
- **Files:** `src/app.js`, `server.js`, updated `package.json`
- **Commit message:** `feat: setup Express app with core middleware and health check`

**Commit 5: Setup database migrations framework**

- **What to do:**
    - Install `node-pg-migrate`
    - Create migration config file
    - Create initial migration based on `schema.sql`:
        - Create users table

        - Create stalls table
        - Create literary_genres table
        - Create reservations table
        - Add indexes
    - Add npm script to run migrations
- **Files:** `migrations/001_initial_schema.sql`, migration config, updated `package.json`
- **Commit message:** `feat: setup database migrations with initial schema`

---

## Phase 2: Authentication & User Management (Commits 6-9)

### Commit 6: Create User model and authentication utilities

- **What to do:**
    - Create `src/models/User.js` with methods:
        - `findByEmail(email)`
        - `findById(id)`
        - `create(userData)`
        - `updatePassword(id, password)`
    - Create `src/utils/hash.js` with bcrypt hash/compare functions
    - Create `src/utils/jwt.js` with sign/verify JWT functions
- **Files:** `src/models/User.js`, `src/utils/hash.js`, `src/utils/jwt.js`
- **Commit message:** `feat: create User model and auth utilities`

### Commit 7: Implement user registration endpoint

- **What to do:**
    - Create `src/controllers/authController.js` with `register` function
    - Validate input (email, password, name, role)
    - Hash password before saving
    - Create `src/routes/authRoutes.js` with `POST /api/auth/register`
    - Add validation middleware
    - Mount routes in `app.js`
- **Files:** `src/controllers/authController.js`, `src/routes/authRoutes.js`, updated `app.js`
- **Commit message:** `feat: implement user registration endpoint`

### Commit 8: Implement user login endpoint

- **What to do:**
    - Add `login` function to `authController.js`
    - Verify email exists and password matches
    - Generate JWT token with user id and role
    - Return token and user info
    - Add `POST /api/auth/login` route
- **Files:** updated `src/controllers/authController.js`, updated `src/routes/authRoutes.js`
- **Commit message:** `feat: implement user login with JWT authentication`

**Commit 9: Create authentication and authorization middleware**

- **What to do:**
  - Create `src/middlewares/auth.js` with:
    - `authenticate` - verify JWT token
    - `authorize(...roles)` - check user role
  - Create `src/middlewares/errorHandler.js` for centralized error handling
  - Add error handler to `app.js`
- **Files:** `src/middlewares/auth.js`, `src/middlewares/errorHandler.js`, updated `app.js`
- **Commit message:** `feat: add authentication and authorization middleware`

---

## Phase 3: Stall Management (Commits 10-12)

**Commit 10: Create Stall model**

- **What to do:**
  - Create `src/models/Stall.js` with methods:
    - `findAll(filters)` - get all stalls with optional filters (size, availability)
    - `findById(id)`
    - `findByName(name)`
    - `create(stallData)` - ADMIN only
    - `update(id, stallData)` - ADMIN only
    - `delete(id)` - ADMIN only
- **Files:** `src/models/Stall.js`
- **Commit message:** `feat: create Stall model with CRUD methods`

**Commit 11: Implement stall CRUD endpoints**

- **What to do:**
  - Create `src/controllers/stallController.js` with:
    - `getAllStalls` - public, with filters
    - `getStallById` - public
    - `createStall` - ADMIN only
    - `updateStall` - ADMIN only
    - `deleteStall` - ADMIN only
  - Create `src/routes/stallRoutes.js`
  - Add validation middleware for stall data
  - Mount routes in `app.js`
- **Files:** `src/controllers/stallController.js`, `src/routes/stallRoutes.js`, updated `app.js`
- **Commit message:** `feat: implement stall CRUD endpoints with RBAC`

**Commit 12: Add stall availability check endpoint**

- **What to do:**
  - Add `checkAvailability` method to Stall model
  - Add `getAvailableStalls` controller function

    - Add `GET /api/stalls/available` route with filters (size, date range)
    - Return only stalls with `is_reserved = false` or no active reservations
- **Files:** updated `src/models/Stall.js`, updated `src/controllers/stallController.js`, updated routes
- **Commit message:** `feat: add stall availability check endpoint`

---

## Phase 4: Core Reservation Logic (Commits 13-18)

### Commit 13: Create Reservation model

- **What to do:**
    - Create `src/models/Reservation.js` with methods:
        - `findAll(filters)` - get reservations with optional filters
        - `findById(id)`
        - `findByUserId(userId)`
        - `findByStallId(stallId)`
        - `findByQRCode(qrCode)`
        - `create(reservationData)` - transactional
        - `cancel(id)` - update status and set cancelled_at
- **Files:** `src/models/Reservation.js`
- **Commit message:** `feat: create Reservation model with query methods`

### Commit 14: Implement QR code generation utility

- **What to do:**
    - Create `src/utils/qrGenerator.js` with:
        - `generateQRCode(reservationId, userId, stallId)` - create unique QR string
        - `generateQRImage(qrString)` - create base64 QR image
    - Use UUIDv4 or combination of timestamp + ids + random hash
- **Files:** `src/utils/qrGenerator.js`
- **Commit message:** `feat: implement QR code generation utility`

### Commit 15: Implement reservation creation with transaction

- **What to do:**
    - Create `src/services/reservationService.js` with `createReservation` function
    - Use database transaction to:
        1. Check stall availability (SELECT FOR UPDATE)
        2. Insert reservation record
        3. Update stall `is_reserved = true`
        4. Generate QR code
    - Handle race conditions with proper locking
- **Files:** `src/services/reservationService.js`
- **Commit message:** `feat: implement transactional reservation creation with locking`

### Commit 16: Create reservation endpoints

- **What to do:**
  - Create `src/controllers/reservationController.js` with:
    - `createReservation` - authenticated users
    - `getUserReservations` - get own reservations
    - `getReservationById` - owner/admin/employee only
    - `getAllReservations` - ADMIN/EMPLOYEE only
  - Create `src/routes/reservationRoutes.js`
  - Add validation middleware
  - Mount routes in `app.js`
- **Files:** `src/controllers/reservationController.js`, `src/routes/reservationRoutes.js`, updated `app.js`
- **Commit message:** `feat: implement reservation creation and retrieval endpoints`

**Commit 17: Implement reservation cancellation**

- **What to do:**
  - Add `cancelReservation` function to reservationService.js
  - Use transaction to:
    1. Update reservation status to CANCELLED
    2. Set cancelled_at timestamp
    3. Update stall `is_reserved = false`
  - Add `cancelReservation` controller function
  - Add `DELETE /api/reservations/:id` route
  - Only owner/admin can cancel
- **Files:** updated `src/services/reservationService.js`, updated controller, updated routes
- **Commit message:** `feat: implement reservation cancellation with rollback`

**Commit 18: Add reservation validation and business rules**

- **What to do:**
  - Create `src/middlewares/reservationValidator.js`
  - Add validation rules:
    - User can't have multiple active reservations for same stall
    - Stall must exist and be available
    - User must have valid role
  - Add custom error classes for reservation errors
- **Files:** `src/middlewares/reservationValidator.js`, `src/utils/errors.js`
- **Commit message:** `feat: add reservation validation and business rules`

---

## Phase 5: Literary Genres & Advanced Features (Commits 19-21)

**Commit 19: Create Literary Genres endpoints**

- **What to do:**
  - Create `src/models/LiteraryGenre.js`
  - Create `src/controllers/genreController.js`

    - Add endpoints:
        - `POST /api/genres` - add genre for user
        - `GET /api/genres/user/:userId` - get user genres
        - `DELETE /api/genres/:id` - delete genre
    - Create `src/routes/genreRoutes.js`
- **Files:** `src/models/LiteraryGenre.js`, `src/controllers/genreController.js`, `src/routes/genreRoutes.js`
- **Commit message:** `feat: implement literary genres management`

**Commit 20: Add reservation search and filtering**

- **What to do:**
    - Enhance `getAllReservations` with query params:
        - Filter by status, date range, user_id, stall_id
        - Pagination support (limit, offset)
        - Sorting options
    - Add `GET /api/reservations/search` endpoint
- **Files:** updated `src/controllers/reservationController.js`, updated routes
- **Commit message:** `feat: add advanced reservation search and filtering`

**Commit 21: Implement QR code verification endpoint**

- **What to do:**
    - Create `src/services/qrService.js` with `verifyQRCode` function
    - Add `POST /api/reservations/verify` endpoint
    - Accept QR code string, return reservation details if valid
    - Mark reservation as "checked-in" (optional status update)
    - EMPLOYEE/ADMIN only
- **Files:** `src/services/qrService.js`, updated controller, updated routes
- **Commit message:** `feat: implement QR code verification for check-in`

---

## Phase 6: Testing & Quality (Commits 22-23)

**Commit 22: Add unit tests for models and services**

- **What to do:**
    - Create `tests/unit/` folder
    - Write tests for:
        - User model (findByEmail, create)
        - Stall model (availability check)
        - Reservation service (transaction logic)
        - QR generator (uniqueness)
    - Setup Jest config
    - Add npm script: `npm test`
- **Files:** `tests/unit/*.test.js`, `jest.config.js`, updated `package.json`
- **Commit message:** `test: add unit tests for models and services`

**Commit 23: Add integration tests for reservation flow**

- **What to do:**
  - Create `tests/integration/` folder
  - Write end-to-end tests using Supertest:
    - User registration and login
    - Stall listing and availability
    - Reservation creation (happy path)
    - Concurrent reservation attempts (race condition)
    - Reservation cancellation
  - Setup test database configuration
- **Files:** `tests/integration/*.test.js`, `tests/setup.js`
- **Commit message:** `test: add integration tests for reservation workflows`

---

## Phase 7: Documentation & Deployment (Commits 24-25)

**Commit 24: Add API documentation and logging**

- **What to do:**
  - Install `swagger-jsdoc` and `swagger-ui-express`
  - Create `src/config/swagger.js`
  - Add JSDoc comments to all routes
  - Add Swagger UI at `GET /api-docs`
  - Implement request/response logging with Winston/Pino
  - Create `src/config/logger.js`
- **Files:** `src/config/swagger.js`, `src/config/logger.js`, JSDoc comments in routes, updated `app.js`
- **Commit message:** `docs: add Swagger API documentation and logging`

**Commit 25: Production configuration and deployment guide**

- **What to do:**
  - Create `ecosystem.config.js` for PM2
  - Add production environment variables to `.env.example`
  - Create `Dockerfile` and `docker-compose.yml`
  - Update `README.md` with:
    - Setup instructions
    - API endpoints list
    - Environment variables
    - Migration commands
    - Testing commands
    - Deployment steps (Docker/PM2)
  - Add health check improvements
- **Files:** `ecosystem.config.js`, `Dockerfile`, `docker-compose.yml`, updated `README.md`
- **Commit message:** `chore: add production config and deployment documentation`

---

## Quick Start After All Commits

```
# Install dependencies
npm install

# Setup environment
cp .env.example .env
# Edit .env with your PostgreSQL credentials

# Run migrations
npm run migrate

# Start development server
npm run dev

# Run tests
npm test

# Build for production
npm start

# View API docs
# Navigate to http://localhost:3000/api-docs
```

---

## Database Migration Enhancement (Optional)

After Commit 5, you may want to add a migration to include:

- Foreign key constraints (reservations → users, reservations → stalls, literary_genres → users)
- Unique partial index for active reservations: `CREATE UNIQUE INDEX idx_active_reservation_per_stall ON reservations(stall_id) WHERE status = 'CONFIRMED';`

This can be **Commit 5.5** or folded into Commit 15.

---

## Next Steps After Completion

1. **Integration with Frontend:** Connect React portals (employee-portal, publisher-portal)
2. **Notification Service:** Send email/SMS on reservation confirmation
3. **Payment Integration:** Add payment gateway for stall fees
4. **Analytics Dashboard:** Reservation metrics, revenue tracking
5. **Rate Limiting:** Prevent abuse with express-rate-limit
6. **Caching:** Add Redis for frequently accessed data

---

## Notes

- Each commit should be independently testable
- Run migrations and tests after each commit
- Keep commits atomic and focused on one feature
- Write meaningful commit messages
- Update README.md as you add new features

---

**Ready to start?** Ask me for the code for **Commit 1** and I'll provide the complete implementation!