Core Application Architecture

Relevant source files

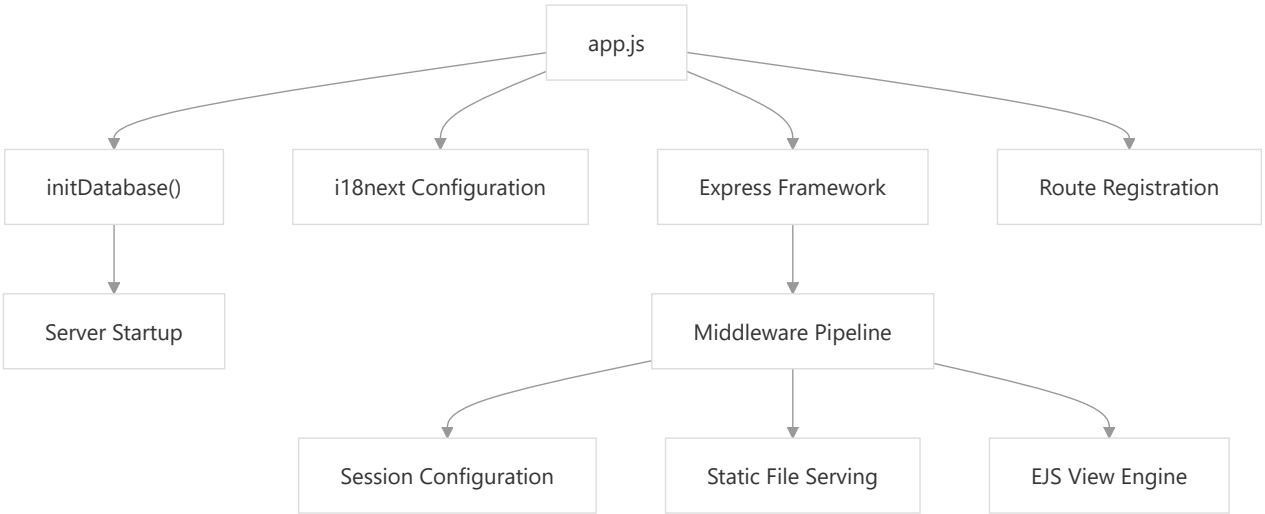
This document describes the core Express.js application structure that forms the foundation of PadelFlow, including server configuration, routing architecture, middleware pipeline, and key application services. It covers the main `app.js` entry point and how various components are wired together to create the web application framework.

For information about specific user roles and authentication mechanisms, see User Roles and Authentication. For details about database layer components, see Database Layer. For development environment setup, see Development Environment.

Express Application Structure

PadelFlow is built on Node.js using the Express.js web framework. The main application entry point is defined in `app.js`, which orchestrates the entire server setup and configuration.

Core Application Initialization



Application Bootstrap Process

Sources: app.js 1-78

The application follows a standard Express.js initialization pattern with several key configuration steps:

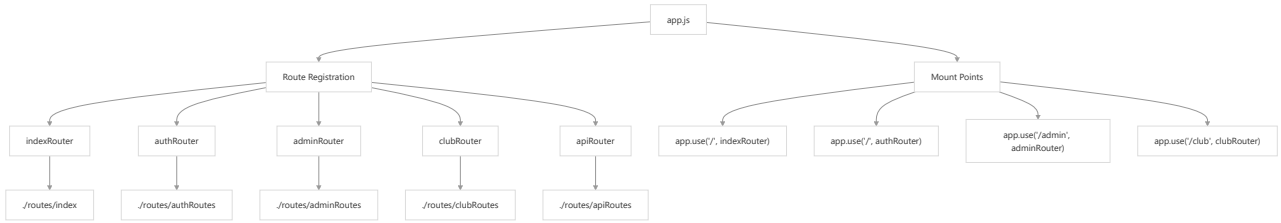
Configuration Step	Purpose	Implementation
Express Setup	Core web framework initialization	`express()`
Middleware Stack	Request processing pipeline	`express.urlencoded()` , `express.json()` , `express.static()`
Session Management	User state persistence	`express-session` with cookie configuration

Configuration Step	Purpose	Implementation
View Engine	Server-side rendering	EJS templating engine
Internationalization	Multi-language support	i18next with filesystem backend
Database Connection	Data persistence layer	SQLite via `initDatabase()`
Route Registration	HTTP endpoint definitions	Multiple route modules

Routing Architecture

The application uses a modular routing structure where different functional areas are separated into dedicated route files.

Route Module Organization



Route Module Registration

Sources: app.js 10-16 app.js 62-67

The routing system is organized into the following modules:

Route Module	Mount Path	Purpose	Status
`indexRouter`	`/`	Landing page and public routes	Active
`authRouter`	`/`	Authentication endpoints	Active
`adminRouter`	`/admin`	Platform administration	Active
`clubRouter`	`/club`	Club management functionality	Active
`apiRouter`	`/`	API endpoints and real-time features	Commented out

Middleware Pipeline

The Express middleware stack processes all incoming requests through a series of functions that handle various concerns like parsing, authentication, and localization.

Middleware Execution Flow

Middleware Configuration

Sources: `app.js` 36-59

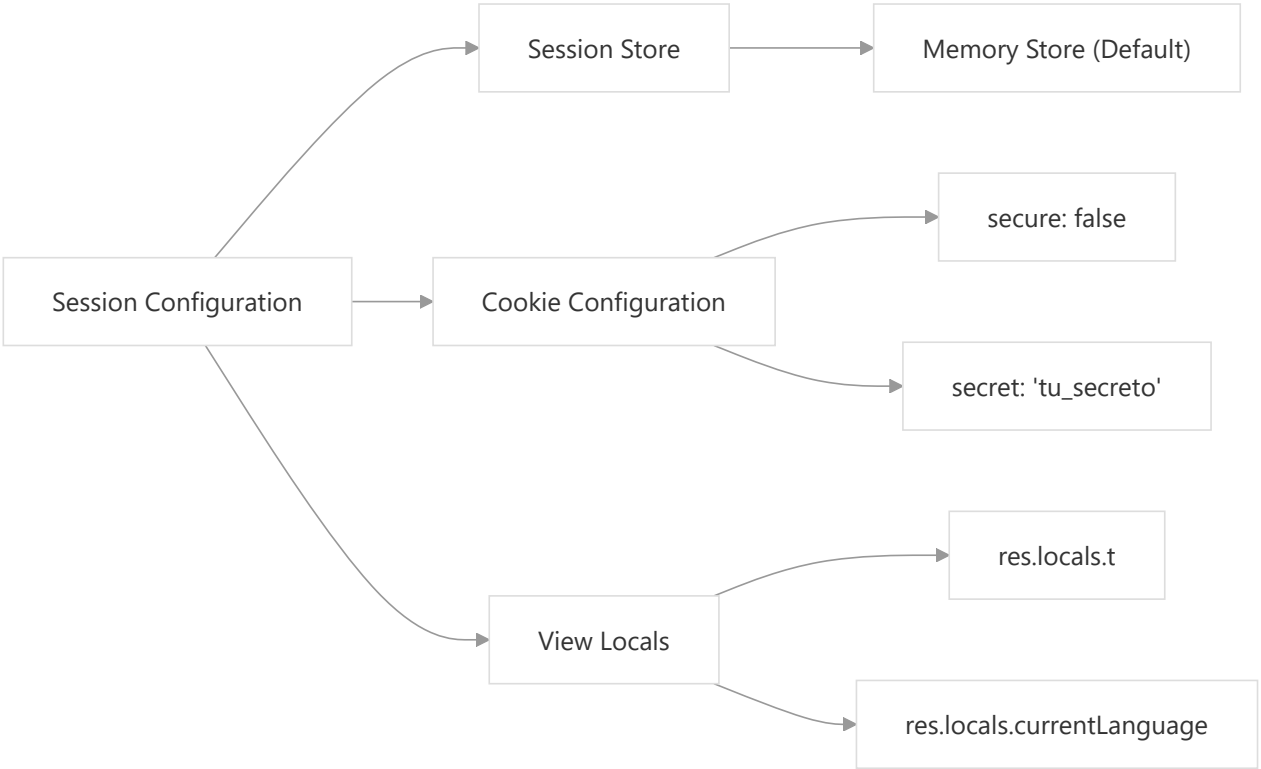
The middleware pipeline includes the following components in execution order:

1. **Body Parsing**: `express.urlencoded({ extended: true })` and `express.json()` handle request body parsing
2. **Static Files**: `express.static('public')` serves static assets from the public directory
3. **Internationalization**: `i18nextMiddleware.handle(i18next)` processes language detection and translation
4. **Session Management**: `session()` middleware with cookie-based session storage
5. **View Locals**: Custom middleware that exposes `req.t` and `req.language` to EJS templates

Session and State Management

The application uses Express sessions for maintaining user state across HTTP requests.

Session Configuration



Session Settings

Sources: app.js 47-52

The session configuration includes:

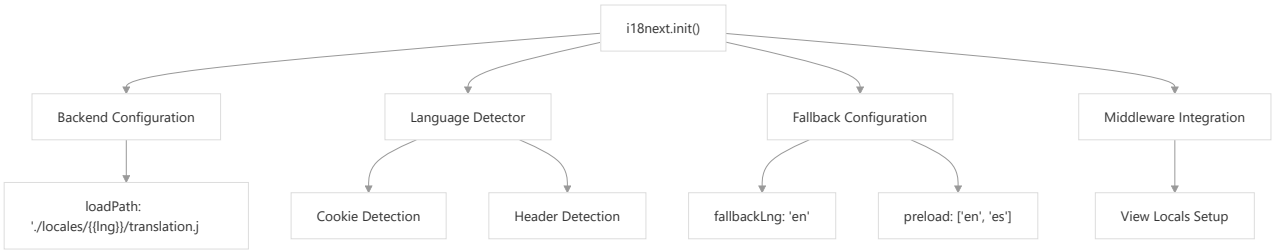
Setting	Value	Purpose
`secret`	`'tu_secreto'`	Session signing key (should be environment variable)
`resave`	`false`	Prevents unnecessary session saves

Setting	Value	Purpose
`saveUninitialized`	`true`	Saves new sessions even if unmodified
`cookie.secure`	`false`	Allows cookies over HTTP (set to true for HTTPS)

Internationalization System

PadelFlow includes comprehensive internationalization support using the i18next library with filesystem-based translation storage.

i18next Configuration



Internationalization Features

Sources: app.js 17-31 app.js 40-41 app.js 55-58

The i18next configuration provides:

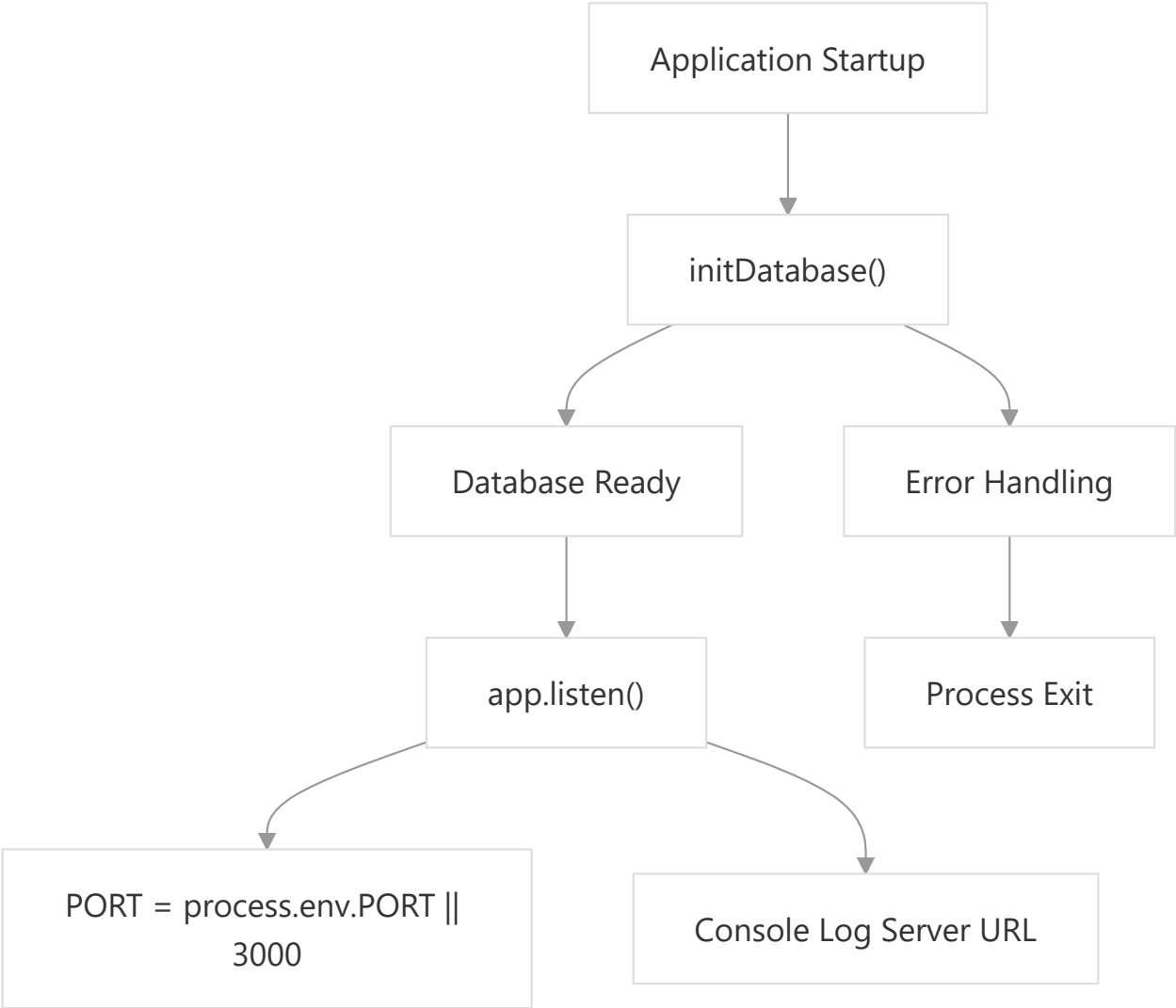
- **Backend**: Filesystem backend loading from `./locales/{{lng}}/translation.json`
- **Language Detection**: Automatic detection via cookies and HTTP headers
- **Fallback Language**: English (`en`) as the default fallback

- **Preloaded Languages**: English and Spanish supported out of the box
- **Development Features**: `saveMissing: true` helps identify missing translation keys
- **View Integration**: `req.t` translation function and `req.language` available in all EJS templates

Database Integration

The application initializes the SQLite database before starting the HTTP server, ensuring data persistence is available for all requests.

Database Initialization Flow



Database Startup Process

Sources: app.js 6 app.js 70-78

The database initialization follows this pattern:

1. Import `initDatabase` function from `./db/database`
2. Call `initDatabase()` which returns a Promise
3. On success, start the HTTP server on the configured port
4. On failure, log error and exit process
5. Server listens on `process.env.PORT` or default port 3000

View Engine Configuration

The application uses EJS (Embedded JavaScript) as its templating engine for server-side rendering.

Sources: app.js 44

The view engine is configured with `app.set('view engine', 'ejs')`, enabling the rendering of `.ejs` template files from the `views` directory. Templates have access to session data, translation functions, and request context through the middleware pipeline.