



Last indexed: 27 August 2025 (c12f7a)

PadelFlow Overview

Core Application Architecture

Server Setup and Configuration

User Roles and Authentication

Tournament Management
Features

Real-time Features

Database Layer

SQLite Database Management

Database Extensions

Qt Framework Components

Image Format Support

Development Environment

IDE Configuration

Debugging Setup

Project Configuration

Server Setup and Configuration

Relevant source files

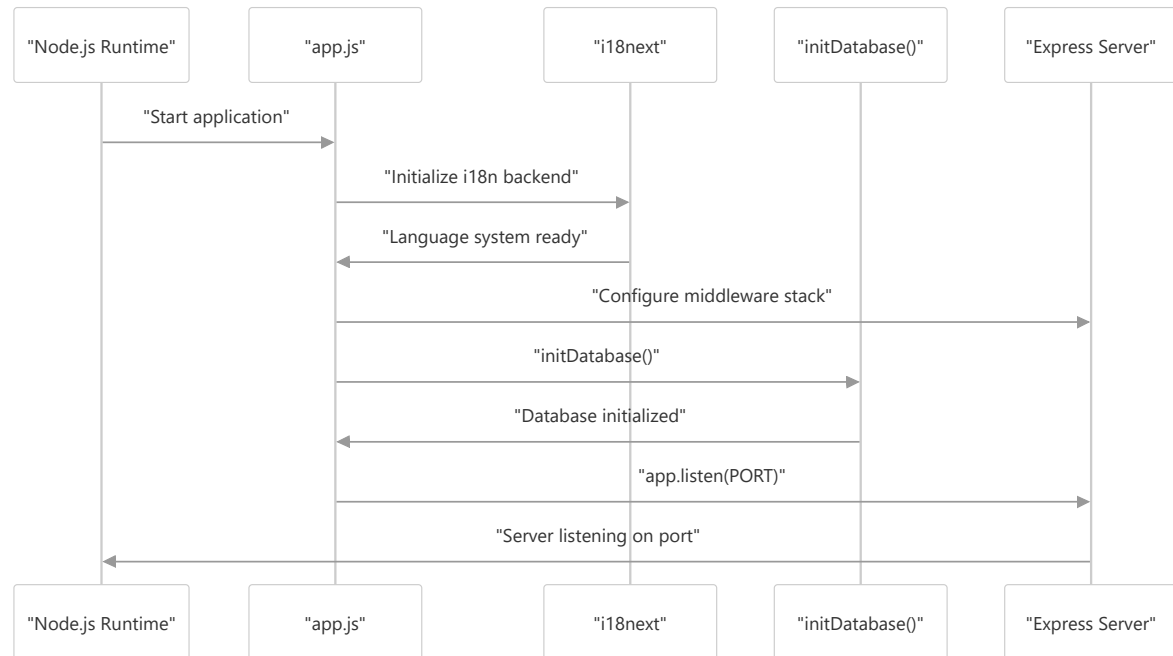
This document covers the core Express.js server configuration, middleware setup, and application initialization process for PadelFlow. It explains how the main server application is bootstrapped, configured with internationalization support, session management, and routing infrastructure.

For information about user authentication and authorization middleware, see [User Roles and Authentication](#). For real-time communication setup, see [Real-time Features](#). For database schema and extensions, see [Database Layer](#).

Application Bootstrap Process

The PadelFlow server follows a standard Express.js application structure with enhanced internationalization and session management capabilities. The main application entry point coordinates database initialization, middleware configuration, and route registration.

Server Initialization Flow

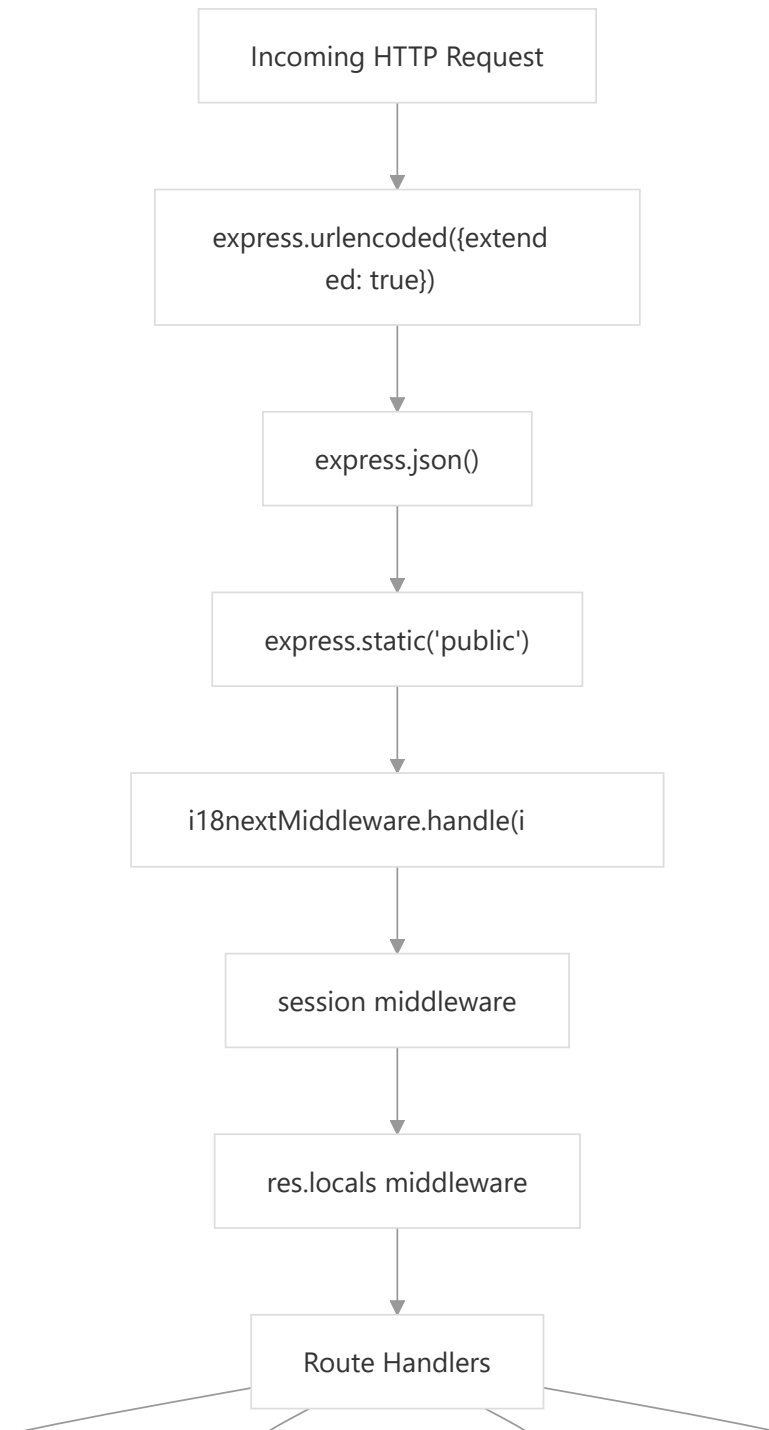


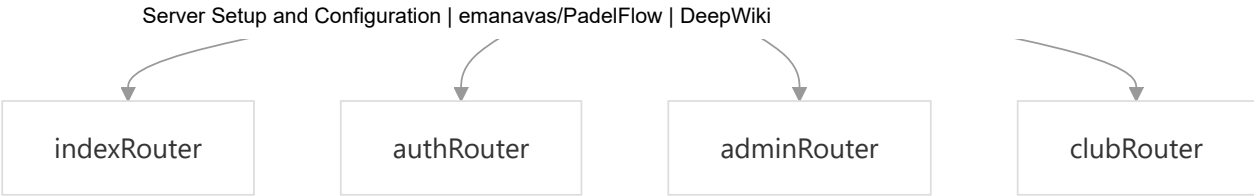
Sources:  `app.js` | 72-78

Express Application Configuration







The main Express application is configured with several key middleware components that provide core functionality for request processing, session management, and internationalization.


Core Middleware Stack





The middleware stack processes requests in a specific order, with each component adding functionality:

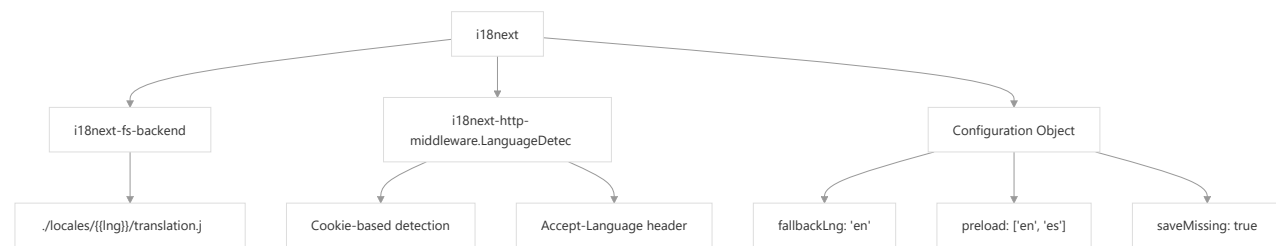
Middleware	Purpose	Configuration Location	
<code>express.urlencoded</code>	Parse form data	 <code>app.js</code>	36
<code>express.json</code>	Parse JSON request bodies	 <code>app.js</code>	37
<code>express.static</code>	Serve static files from <code>public/</code>	 <code>app.js</code>	38
<code>i18nextMiddleware.handle</code>	Process language preferences	 <code>app.js</code>	41
<code>session</code>	Manage user sessions	 <code>app.js</code>	47-52
Custom locals middleware	Expose translation functions	 <code>app.js</code>	55-59

Sources:  `app.js` | 35-59

Internationalization System

PadelFlow implements comprehensive internationalization support using the `i18next` library with file-based translation storage and automatic language detection.

i18next Configuration



The internationalization system supports:

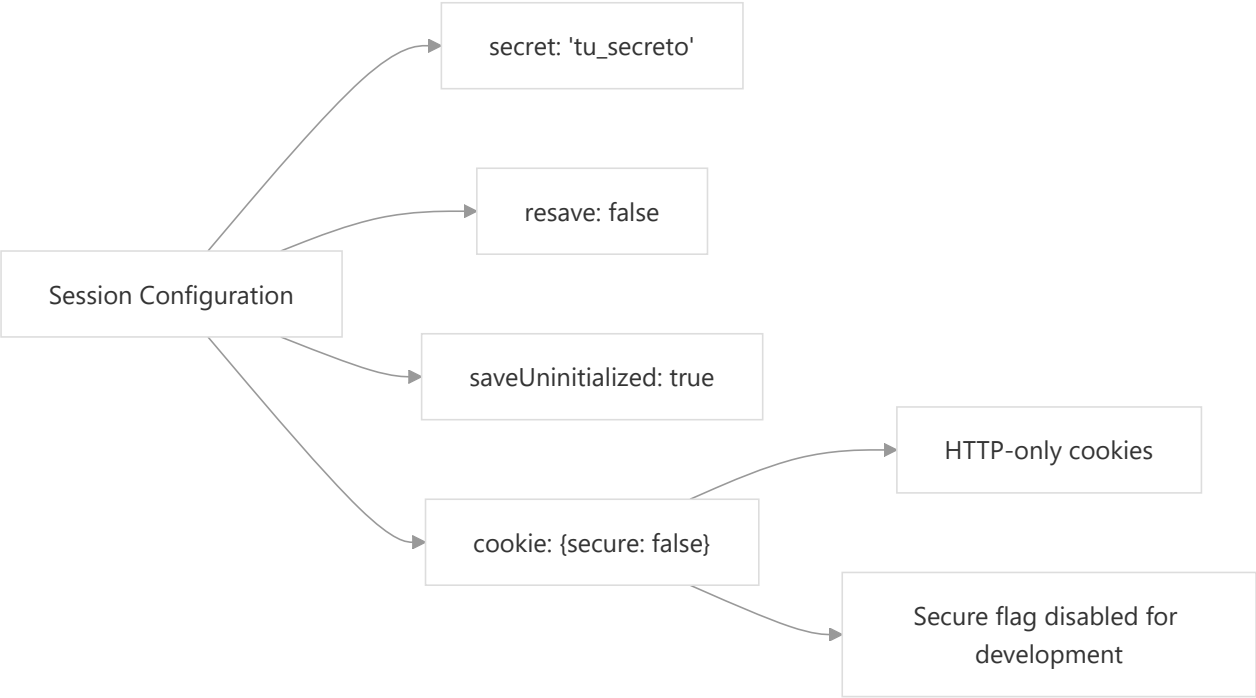
- **Backend:** File-system based translations stored in `/locales/{{lng}}/translation.json`
- **Language Detection:** Cookie-first, then HTTP headers
- **Fallback:** English (`en`) as default language
- **Preloaded Languages:** English and Spanish
- **Development Features:** Automatic missing key detection

Sources:  `app.js` | 17-31

Session Management

The application uses `express-session` middleware for managing user authentication state and session data across requests.

Session Configuration



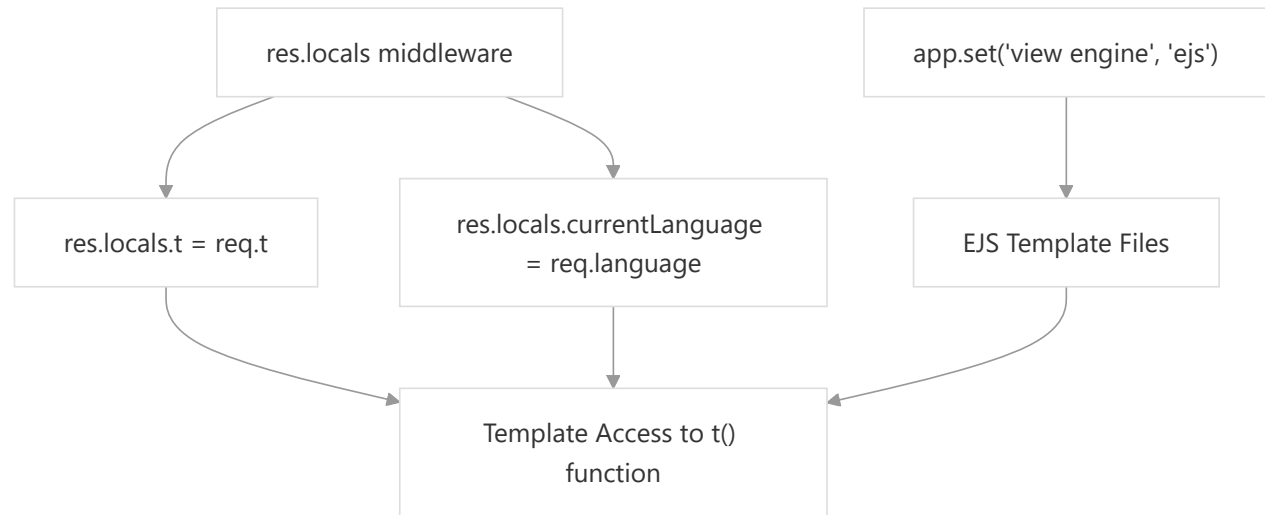
Configuration	Value	Purpose
secret	'tu_secreto'	Session encryption key (should use environment variable in production)
resave	false	Don't save session if unmodified
saveUninitialized	true	Save new but unmodified sessions
cookie.secure	false	Allow cookies over HTTP (development setting)

Sources:  app.js | 47-52

View Engine and Template Configuration

The application uses EJS (Embedded JavaScript) as the template engine for server-side rendering of HTML pages.

Template System Setup



The template configuration provides:

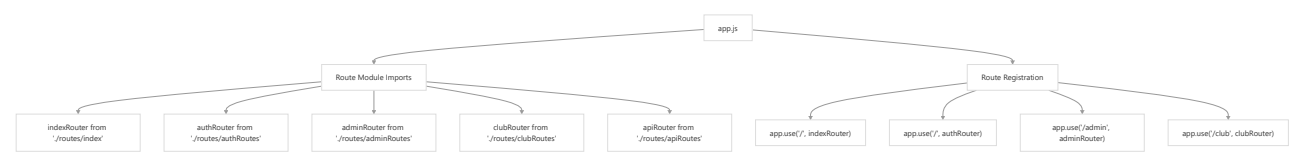
- **Template Engine:** EJS for server-side rendering
- **Translation Access:** `res.locals.t` function available in all templates
- **Language Context:** `res.locals.currentLanguage` for language-aware rendering

Sources:  `app.js` | 44  `app.js` | 55-59

Route Registration

The application organizes functionality into separate route modules that handle different aspects of the system.

Route Module Architecture



Route Module	Mount Path	Purpose
indexRouter	/	Homepage and general routes
authRouter	/	Authentication endpoints
adminRouter	/admin	Administrative functionality
clubRouter	/club	Club management features
apiRouter	/	API endpoints (commented out)

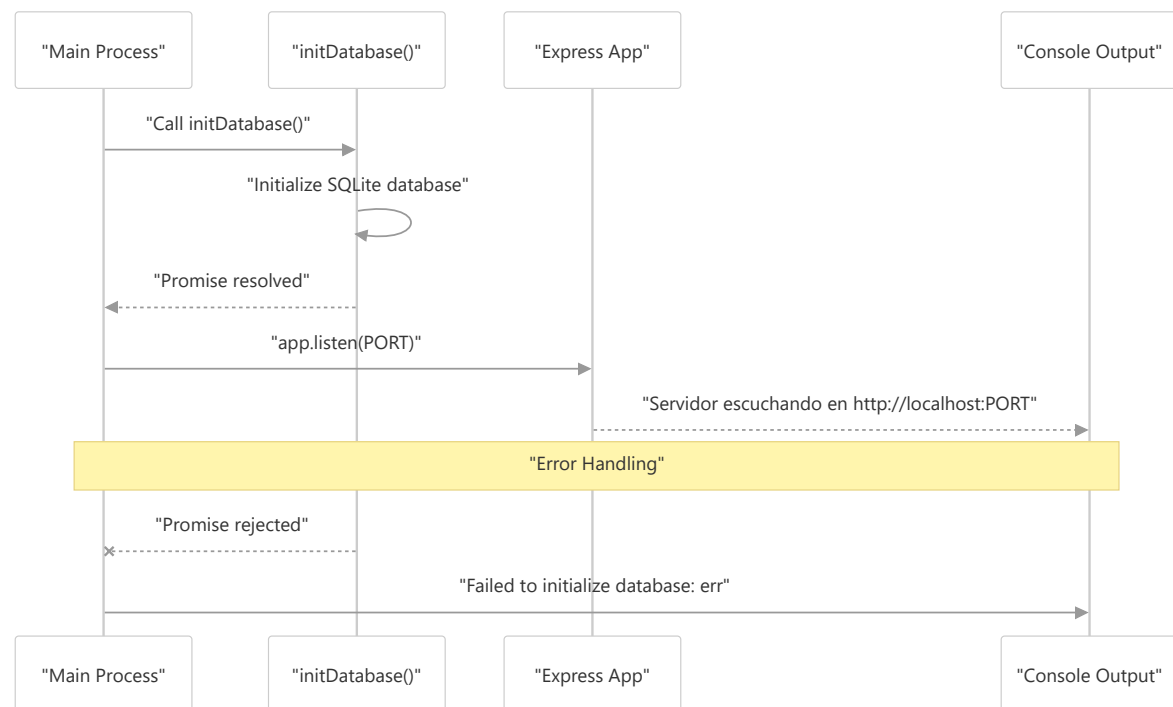
Note that `tournamentRouter` and `apiRouter` are imported but not currently mounted in the application.

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

Server Startup Process

The application implements a robust startup sequence that ensures database connectivity before accepting HTTP requests.

Startup Sequence



The startup process:

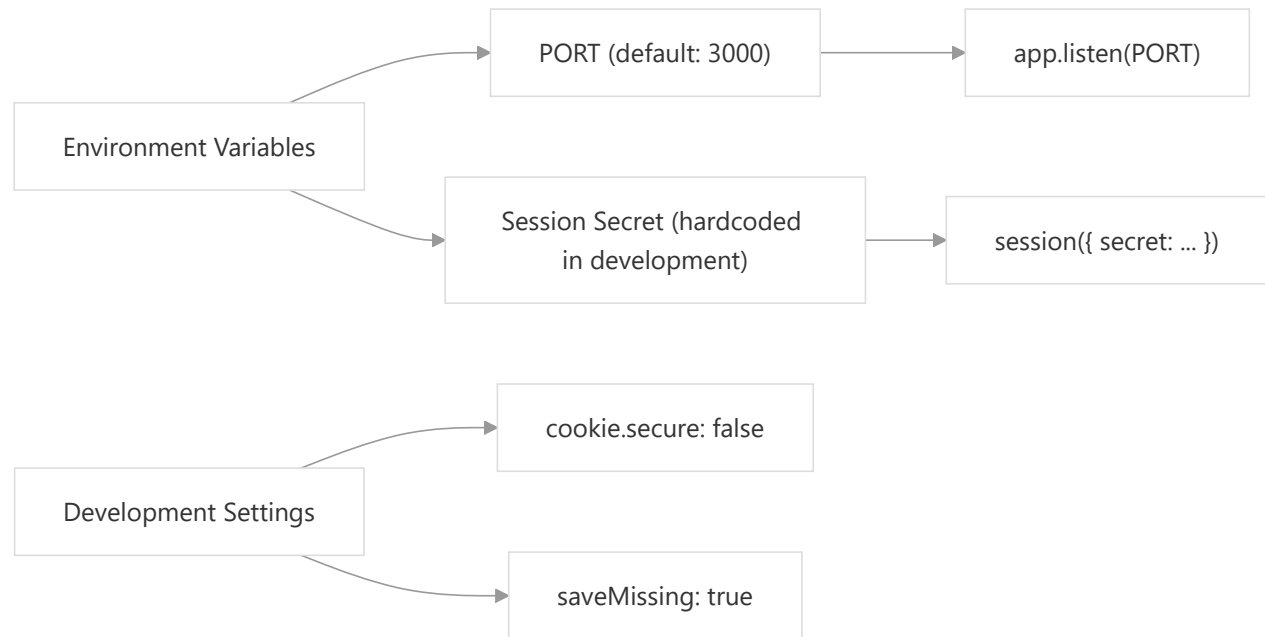
1. **Database Initialization:** Calls `initDatabase()` and waits for completion
2. **Port Configuration:** Uses `process.env.PORT` or defaults to `3000`
3. **Server Start:** Only starts HTTP server after successful database initialization
4. **Error Handling:** Logs database initialization failures and prevents server startup

Sources:  `app.js` | 69-78

Environment Configuration

The server supports environment-based configuration for production deployment while providing sensible defaults for development.

Configuration Variables



Current configuration approach:

- **Port:** Uses `process.env.PORT` with fallback to `3000`
- **Session Secret:** Hardcoded string (should be environment variable in production)
- **Cookie Security:** Disabled for HTTP development
- **i18n Development:** Missing translation key detection enabled

Sources:  app.js | 48  app.js | 51  app.js | 70