INTRODUCTION TO EXPRESS.JS

WHAT IS EXPRESS.JS?

Express.js is a fast, unopinionated, minimalist web framework for Node.js.

It simplifies building server-side applications by providing powerful tools for routing, middleware integration, and request handling.

Often used to build web applications, RESTful APIs, and single-page applications.

WHY USE EXPRESS.JS?

1. Lightweight and Flexible

Express.js provides core functionalities without unnecessary overhead, allowing developers to structure their applications freely and scale them for projects of any size. Its minimalist design ensures efficiency and adaptability.

2. Large Ecosystem of Middleware

With a wide range of middleware, Express enables easy integration of features like authentication, logging, security, and data parsing, significantly speeding up development.

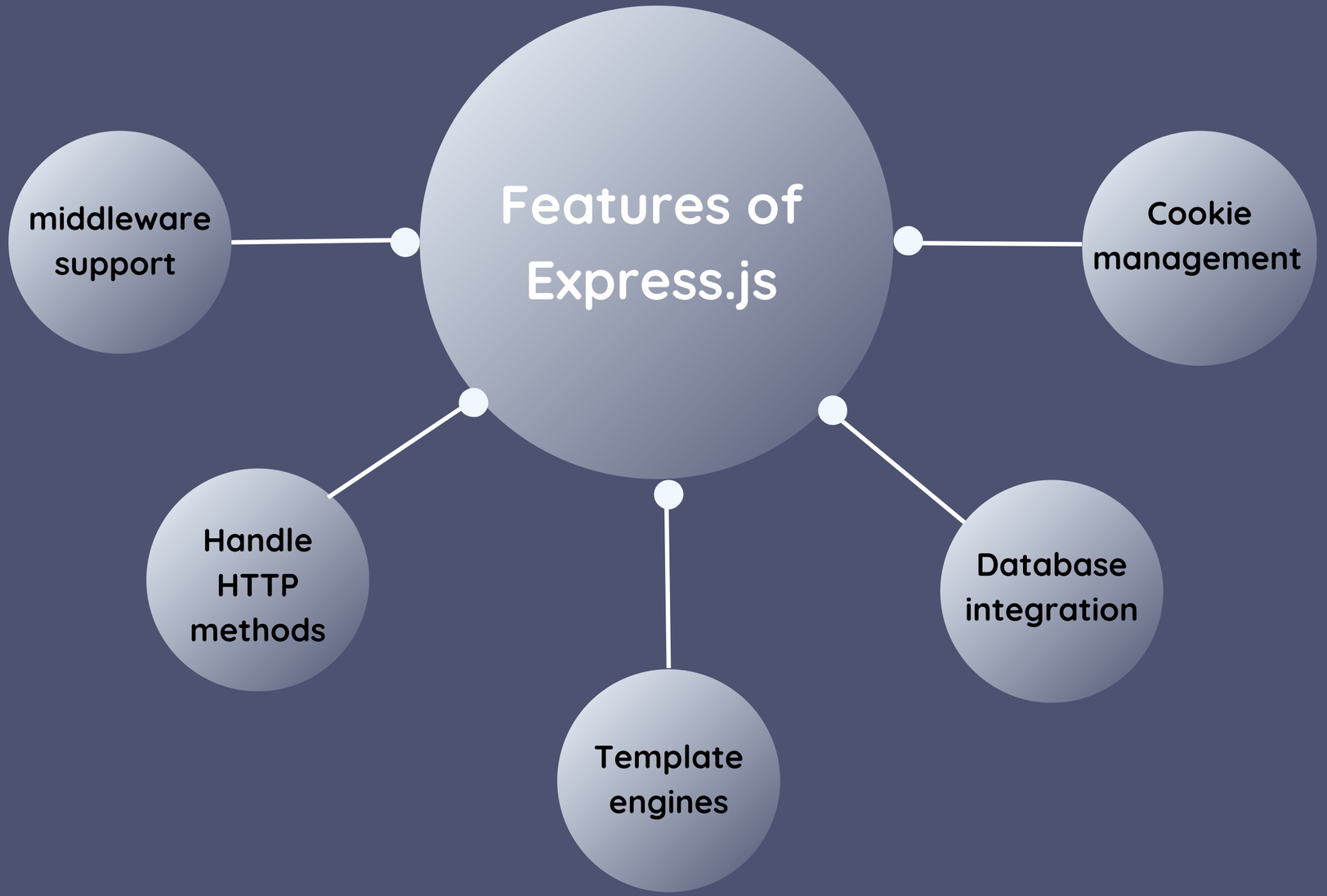
3. Full Control Over the Request-Response Cycle

Developers can customize every aspect of how requests are handled and responses are sent, making Express ideal for building tailored APIs and complex workflows.

4. Active Community and Extensive Documentation

Express benefits from an active developer community, providing rich documentation, support forums, and a constant stream of tools and updates to keep the framework modern and reliable.

KEY FEATURES OF EXPRESS.JS



SETTING UP EXPRESS.JS

Prerequisites

- Install Node.js.

- Install a code editor (e.g., Visual Studio Code).

Steps

1.Initialize a Project:

```
mkdir my-express-app
cd my-express-app
npm init -y
```

2.Install Express:

```
npm install express
```

3.Create a Server.js

```
const express = require('express');
const app = express();
app.get('/', (req, res) => {
    res.send('Hello, World!');
});
app.listen(3000, () => {
    console.log('Server is running on http://localhost:3000');
});
```

4.Run the Server:

```
node server.js
```

CORE FEATURES OF EXPRESS.JS

Routing

Define routes to handle different URLs.

Example:

```
app.get('/about', (req, res) => {
    res.send('About Page');
});
```

Middleware

Functions that execute during the request-response cycle.

Example:

```
app.use((req, res, next) => {
    console.log('Request received');
    next();
});
```

CORE FEATURES OF EXPRESS.JS

Template Engines

Render dynamic HTML pages using engines like EJS, Pug, or Handlebars.
Example with EJS:
npm install ejs
javascript code:

```
app.set('view engine', 'ejs');
app.get('/', (req, res) => {
    res.render('index', { title: 'Home Page' });
});
```

Satic Files:

Serve CSS, images, or JavaScript files.
Example:

```
app.use(express.static('public'));
```

EXPRESS.JS ADVANTAGES

- Minimalistic: Focused on web development without unnecessary complexity.
- Customizable: Add middleware and routes as needed.
- Speed: Optimized for performance and fast response times.
- Community: Large and active community for support and resources.