3. Express.js and Routing

Express.js is a popular web framework for Node.js that simplifies the process of building robust and scalable web applications. Below is a simple example of a RESTful API using Express.js to perform CRUD operations on a resource, let's say, a collection of books.

```javascript
Const express = require('express');
Const bodyParser = require('body-parser');

Const app = express();
Const port = 3000;

App.use(bodyParser.json());

// Sample data for books
Let books = [
  { id: 1, title: 'The Great Gatsby', author: 'F. Scott Fitzgerald' },
  { id: 2, title: 'To Kill a Mockingbird', author: 'Harper Lee' },
];

// Route for getting all books
App.get('/books', (req, res) => {
  Res.json(books);
});

// Route for getting a specific book by ID
App.get('/books/:id', (req, res) => {
  Const bookId = parseInt(req.params.id);
  Const book = books.find((b) => b.id === bookId);
```

```
  If (book) {
    Res.json(book);
  } else {
    Res.status(404).json({ message: ‘Book not found’ });
  }
});

// Route for creating a new book
App.post(‘/books’, (req, res) => {
  Const newBook = req.body;
  Books.push(newBook);
  Res.status(201).json(newBook);
});

// Route for updating a book by ID
App.put(‘/books/:id’, (req, res) => {
  Const bookId = parseInt(req.params.id);
  Const updatedBook = req.body;
  Const index = books.findIndex((b) => b.id === bookId);

  If (index !== -1) {
    Books[index] = updatedBook;
    Res.json(updatedBook);
  } else {
    Res.status(404).json({ message: ‘Book not found’ });
  }
});
```

```
// Route for deleting a book by ID
App.delete('/books/:id', (req, res) => {
  Const bookId = parseInt(req.params.id);
  Books = books.filter((b) => b.id !== bookId);
  Res.json({ message: 'Book deleted successfully' });
});

// Start the server
App.listen(port, () => {
  Console.log(`Server is running on http://localhost:${port}`);
});
```

Explanation of Routing in Web Applications:

1. **Endpoint Definition:** Routes in Express.js define the endpoints or URLs that your web application will respond to. In the example above, we have defined routes for getting all books (`/books`), getting a specific book by ID (`/books/:id`), creating a new book (`/books` with POST), updating a book by ID (`/books/:id` with PUT), and deleting a book by ID (`/books/:id` with DELETE).

2. **HTTP Methods:** Routes are associated with HTTP methods (GET, POST, PUT, DELETE) that determine the type of operation to perform on the specified resource. For example, the route `app.get('/books', …)` handles HTTP GET requests for the `/books` endpoint.

3. **Parameters and Middleware:** Routes can include parameters (e.g., `:id` in `/books/:id`) to capture dynamic values from the URL. Middleware functions (e.g., `bodyParser.json()`) can be used to process data before it reaches the route handler.

4. **Route Handlers:** Route handlers are functions that execute when a specific route is matched. In the example, the route handlers handle CRUD operations on the `books` resource.

Routing in web applications helps organize and structure the code, making it more modular and maintainable. Each route can be seen as a separate endpoint responsible for a specific functionality, and the overall application logic is divided into manageable pieces.