Code Explanation: src\utils\

Table of Contents

constants.js

```
// Explanation modes
exports.MODE_LINE_BY_LINE = 'linebyline';
exports.MODE_FLOWCHART = 'flowchart';
exports.MODE_EXPLAIN = 'explain';
exports.MODE_ARCHITECTURE = 'architecture';
exports.MODE_ARCH = 'arch';  // Alias for architecture
exports.MODE_ISSUES = 'issues';
exports.MODE_ONBOARDING = 'onboarding';

// Output formats
exports.OUTPUT_PDF = 'pdf';
exports.OUTPUT_HTML = 'html';

// Default configuration values
exports.DEFAULT_RETRY_ATTEMPTS = 3;
exports.DEFAULT_CONCURRENCY = 3;
exports.MAX_CONCURRENCY = 10;
exports.MAX_FILE_SIZE_BYTES = 10 * 1024 * 1024; // 10MB default
exports.MAX_FILE_SIZE_MB = 10;
```

AI Explanation

Explanation of `constants.js`

This JavaScript file, located at `D:\MyDev\CodeExplain\src\utils\constants.js`, is designed to define a set of constant values that can be utilized throughout an application. Constants are useful for managing values that are used in multiple places in an application, helping maintainability and readability by avoiding "magic numbers" and hard-coded strings.

Code Breakdown

Explanation Modes

These constants represent different modes of explanation that can be used in the application. Each mode is associated with a specific explanation style or format.

- `exports.MODE_LINE_BY_LINE = 'linebyline';`
  - This mode suggests that the explanation will go through the code line by line, detailing each line's purpose.
- `exports.MODE_FLOWCHART = 'flowchart';`
  - This mode implies that the explanation will present the code structure or logic as a flowchart, which can help visualize decision points and flow.
- `exports.MODE_EXPLAIN = 'explain';`
  - This mode appears to be a general explanation mode, perhaps providing descriptions without a specific structured approach.
- `exports.MODE_ARCHITECTURE = 'architecture';`
  - This mode indicates an explanation focused on the overall design and architecture of the codebase or program.
- `exports.MODE_ARCH = 'arch'; // Alias for architecture`
  - This is an alias for `MODE_ARCHITECTURE`, offering a shorthand way to reference the same concept.
- `exports.MODE_ISSUES = 'issues';`
  - This mode may involve an explanation centering on identified issues within the code, potentially for troubleshooting or debugging.
- `exports.MODE_ONBOARDING = 'onboarding';`
  - This mode suggests a focus on providing explanations suitable for onboarding new developers or users to the codebase.

Output Formats

These constants are used to specify the format of the output generated by the application.

- `exports.OUTPUT_PDF = 'pdf';`
  - The output is expected to be in PDF format.
- `exports.OUTPUT_HTML = 'html';`
  - The output is expected to be in HTML format.

Default Configuration Values

These constants define default configuration settings, possibly for controlling various aspects of application behavior.

- `exports.DEFAULT_RETRY_ATTEMPTS = 3;`
  - This sets the default number of retry attempts to 3, which could be used for network requests or other retryable operations.
- `exports.DEFAULT_CONCURRENCY = 3;`
  - This sets the default concurrency level to 3, which could control the number of concurrent operations or tasks.
- `exports.MAX_CONCURRENCY = 10;`
  - This sets the maximum concurrency level to 10, capping the number of simultaneous operations allowed.
- `exports.MAX_FILE_SIZE_BYTES = 10 * 1024 * 1024; // 10MB default`
  - This sets the maximum file size to 10 megabytes, used for processing or handling files up to this size.
- `exports.MAX_FILE_SIZE_MB = 10;`
  - This simply represents the maximum file size in megabytes, complementing the byte-based limit.

Summary

This `constants.js` file offers a central location for defining various constants that provide consistent reference points for modes, output formats, and default settings across an application. Defining such constants helps avoid repetition, reduce errors, and gives an organized structure to the application configuration.

languageMap.js

```
// Map of file extensions to programming languages
const languageMap = {
  '.js': 'javascript',
  '.jsx': 'javascript',
  '.ts': 'typescript',
  '.tsx': 'typescript',
  '.vue': 'vue',
  '.svelte': 'svelte',
  '.astro': 'astro',
  '.html': 'html',
  '.htm': 'html',
  '.xml': 'xml',
  '.svg': 'xml',
  '.css': 'css',
  '.scss': 'scss',
  '.sass': 'scss',
  '.less': 'less',
  '.php': 'php',
  '.asp': 'asp',
  '.jsp': 'java',
  '.ejs': 'ejs',
  '.hbs': 'handlebars',
  '.handlebars': 'handlebars',
  '.py': 'python',
  '.java': 'java',
  '.c': 'c',
  '.cpp': 'cpp',
  '.cxx': 'cpp',
  '.cc': 'cpp',
  '.h': 'c',
  '.hpp': 'cpp',
  '.cs': 'csharp',
  '.vb': 'vbnet',
  '.fs': 'fsharp',
  '.fsx': 'fsharp',
  '.go': 'go',
  '.rs': 'rust',
  '.swift': 'swift',
  '.kt': 'kotlin',
  '.scala': 'scala',
  '.rb': 'ruby',
  '.pl': 'perl',
  '.pm': 'perl',
  '.tcl': 'tcl',
  '.lua': 'lua',
  '.dart': 'dart',
  '.r': 'r',
  '.m': 'matlab',
  '.matlab': 'matlab',
  '.asm': 'asm',
  '.s': 'asm',
  '.zig': 'zig',
  '.v': 'v',
  '.nim': 'nim',
  '.crystal': 'crystal',
  '.pony': 'pony',
  '.hs': 'haskell',
  '.ml': 'ocaml',
  '.elm': 'elm',
  '.purs': 'purescript',
  '.clj': 'clojure',
  '.cljs': 'clojure',
  '.scm': 'scheme',
  '.rkt': 'racket',
  '.ex': 'elixir',
  '.exs': 'elixir',
  '.erl': 'erlang',
  '.hrl': 'erlang',
  '.graphql': 'graphql',
  '.gql': 'graphql',
  '.sql': 'sql',
  '.json': 'json',
  '.yaml': 'yaml',
  '.yml': 'yaml',
  '.awk': 'awk',
  '.sed': 'sed',
  '.perl': 'perl',
  '.sh': 'bash',
  '.bash': 'bash',
  '.zsh': 'bash',
  '.makefile': 'makefile',
  '.mk': 'makefile',
  '.cmake': 'cmake',
  '.gradle': 'gradle',
  '.maven': 'xml',
  '.pom': 'xml',
  '.dockerfile': 'dockerfile',
  '.containerfile': 'dockerfile',
  '.md': 'markdown',
```

```
  '.markdown': 'markdown',
  '.rst': 'rst',
  '.tex': 'latex',
  '.raku': 'raku',
  '.p6': 'raku',
  '.vala': 'vala',
  '.genie': 'vala',
  '.boo': 'boo',
  '.cobra': 'python',
  '.ahk': 'autohotkey'
};

module.exports = languageMap;
```

AI Explanation

Certainly! Let's break down the contents of the `languageMap.js` JavaScript file.

Explanation of `languageMap.js`

Overview

This JavaScript file contains a single object, `languageMap`, which is a mapping of file extensions to programming languages. It is designed to help identify the programming language based on the file extension of a file. In a typical software development environment, such a mapping can be useful for tools and editors to provide language-specific features like syntax highlighting or linting.

Contents

- **`languageMap` Object**: This object consists of key-value pairs, where the key is a file extension (e.g., `'.js'`) and the value is the corresponding programming language (e.g., `'javascript'`).

Key Components

- **JavaScript and TypeScript**:
  - `'.js'`: Maps to JavaScript
  - `'.jsx'`: Maps to JavaScript (used for React components)
  - `'.ts'`: Maps to TypeScript
  - `'.tsx'`: Maps to TypeScript (used for React components in TypeScript)
- **Web Technologies**:
  - `'.html'` and `'.htm'`: Map to HTML
  - `'.css'`: Maps to CSS
  - Stylesheet pre-processors and variants like `'.scss'` (Sass), `'.less'` (Less)
- **Common Programming Languages**:
  - C-like languages (`'.c'`, `'.cpp'`, `'.h'`, etc.)
  - Java (`'.java'`, `'.jsp'`)
  - Python (`'.py'`)
  - Ruby (`'.rb'`)
  - PHP (`'.php'`)
- **Other Languages and Technologies**:
  - `'.json'`: Maps to JSON
  - `'.sql'`: Maps to SQL
  - `'.yaml'`, `'.yml'`: Map to YAML
  - Others: Includes languages like Go (`'.go'`), Rust (`'.rs'`), Swift (`'.swift'`), and many more.
- **Markup and Markdown**:
  - Markdown (`'.md'`, `'.markdown'`)
  - LaTeX (`'.tex'`)
  - reStructuredText (`'.rst'`)
- **Build and Scripting**:
  - Shell scripting files (`'.sh'`, `'.bash'`, `'.zsh'`)
  - Makefiles (`'.mk'`, `'.makefile'`)

- Dockerfiles (`'.dockerfile'` , `'.containerfile'`)

Export

- **`module.exports = languageMap;`**: This line exports the `languageMap` object, making it accessible in other files using `require` in a Node.js environment.

Use Cases

- **IDE Integration**: Automatically associate files with the appropriate syntax highlighter in a code editor.
- **File Type Identification**: Helps tools categorize files based on the language for documentation, building, or scripting automation.
- **Linting and Formatting**: Enables tools to apply language-specific rules for checking or formatting code.

By utilizing the `languageMap` , developers or tools can easily determine which programming language a file is written in based on its extension, facilitating a variety of development tasks.

utils.js

```
/**
 * Common utility functions for CodeExplain
 */

const path = require('path');
const crypto = require('crypto')

/**
 * Escape HTML special characters
 * @param {string} unsafe - String to escape
 * @returns {string} Escaped string
 */
function escapeHtml(unsafe) {
  return unsafe.replace(/[&<>"']/g, (m) => {
    switch (m) {
      case '&': return '&amp;';
      case '<': return '&lt;';
      case '>': return '&gt;';
      case '"': return '&quot;';
      case "'": return '&#039;';
      default: return m;
    }
  });
}

/**
 * Escape regex special characters
 * @param {string} string - String to escape
 * @returns {string} Escaped string
 */
function escapeRegExp(string) {
  return string.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
}

/**
 * Get file extension from path
 * @param {string} filePath - Path to file
 * @returns {string} File extension
 */
function getFileExtension(filePath) {
  return path.extname(filePath).toLowerCase();
}

/**
 * Get file name without extension
 * @param {string} filePath - Path to file
 * @returns {string} File name without extension
 */
function getFileNameWithoutExtension(filePath) {
  return path.basename(filePath, path.extname(filePath));
}

/**
 * Check if a file should be included based on extension
 * @param {string} filePath - Path to file
 * @param {string[]} allowedExtensions - Allowed extensions
 * @returns {boolean} True if file should be included
 */
function shouldIncludeFile(filePath, allowedExtensions) {
  const ext = getFileExtension(filePath);
  return allowedExtensions.includes(ext);
}

/**
 * Format bytes to human readable format
 * @param {number} bytes - Number of bytes
 * @param {number} decimals - Number of decimal places
 * @returns {string} Formatted string
 */
function formatBytes(bytes, decimals = 2) {
  if (bytes === 0) return '0 Bytes';

  const k = 1024;
  const dm = decimals < 0 ? 0 : decimals;
  const sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB'];

  const i = Math.floor(Math.log(bytes) / Math.log(k));

  return parseFloat((bytes / Math.pow(k, i)).toFixed(dm)) + ' ' + sizes[i];
}

/**
 * Capitalize first letter of string
 * @param {string} string - String to capitalize
 * @returns {string} Capitalized string
 */
function capitalizeFirstLetter(string) {
  return string.charAt(0).toUpperCase() + string.slice(1);
}
```

```javascript
/**
 * Convert camelCase to kebab-case
 * @param {string} string - String to convert
 * @returns {string} Converted string
 */
function camelToKebab(string) {
  return string.replace(/([a-z0-9])([A-Z])/g, '$1-$2').toLowerCase();
}

/**
 * Truncate string with ellipsis
 * @param {string} str - String to truncate
 * @param {number} maxLength - Maximum length
 * @returns {string} Truncated string
 */
function truncateString(str, maxLength) {
  if (str.length <= maxLength) return str;
  if (maxLength <= 3) return '...';
  return str.slice(0, maxLength - 3) + '...';
}

/**
 * Generate unique ID
 * @returns {string} Unique ID
 */
function generateUniqueId() {
  return crypto.randomBytes(8).toString('hex')
}

/**
 * Sleep for specified milliseconds
 * @param {number} ms - Milliseconds to sleep
 * @returns {Promise} Promise that resolves after specified time
 */
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

/**
 * Deep merge two objects
 * @param {Object} target - Target object
 * @param {Object} source - Source object
 * @returns {Object} Merged object
 */
function deepMerge(target, source) {
  if (typeof target !== 'object' || target === null) {
    return source;
  }

  if (typeof source !== 'object' || source === null) {
    return target;
  }

  const output = Array.isArray(target) ? [...target] : { ...target };

  for (const key in source) {
    if (source.hasOwnProperty(key)) {
      if (isObject(source[key])) {
        output[key] = deepMerge(output[key] || {}, source[key]);
      } else {
        output[key] = source[key];
      }
    }
  }

  return output;
}

/**
 * Check if value is object
 * @param {*} item - Value to check
 * @returns {boolean} True if value is object
 */
function isObject(item) {
  return (item && typeof item === 'object' && !Array.isArray(item));
}

/**
 * Format date to readable string
 * @param {Date} date - Date to format
 * @returns {string} Formatted date string
 */
function formatDate(date) {
  return date.toLocaleDateString('en-US', {
    year: 'numeric',
    month: 'long',
    day: 'numeric',
    hour: '2-digit',
    minute: '2-digit'
  });
}
```

```
/**
 * Pluralize word based on count
 * @param {number} count - Count
 * @param {string} singular - Singular form
 * @param {string} plural - Plural form
 * @returns {string} Pluralized word
 */
function pluralize(count, singular, plural) {
  return count === 1 ? singular : plural;
}

/**
 * Convert array to human-readable list
 * @param {Array} array - Array to convert
 * @returns {string} Human-readable list
 */
function arrayToHumanReadableList(array) {
  if (array.length === 0) return '';
  if (array.length === 1) return array[0];
  if (array.length === 2) return array.join(' and ');

  // Create a shallow copy to avoid modifying the original array
  const arrCopy = [...array];
  const last = arrCopy.pop();
  return arrCopy.join(', ') + ', and ' + last;
}

function dd(data) {
  process.stdout.write(JSON.stringify(data, null, 2) + '\n');
  process.exit();
}

module.exports = {
  dd,
  escapeHtml,
  escapeRegExp,
  getFileExtension,
  getFileNameWithoutExtension,
  shouldIncludeFile,
  formatBytes,
  capitalizeFirstLetter,
  camelToKebab,
  truncateString,
  generateUniqueId,
  sleep,
  deepMerge,
  isObject,
  formatDate,
  pluralize,
  arrayToHumanReadableList
};
```

AI Explanation

Certainly! Here's a structured explanation of the `utils.js` code in markdown format:

JavaScript Utility Functions Explained

This JavaScript file, `utils.js`, provides several utility functions that can be used across different parts of an application. These utilities cover a range of common operations, from string manipulation to file handling. Below is a breakdown of each function and its purpose.

Imports

javascript const path = require('path'); const crypto = require('crypto'); ```

- **Node.js Modules**:
  - `path`: Provides utilities for handling and transforming file paths.
  - `crypto`: Used for cryptographic functionality, such as creating unique identifiers.

Utility Functions

1. Escape HTML Special Characters

```
function escapeHtml(unsafe) {
  return unsafe.replace(/[&<>"']/g, (m) => {
    switch (m) {
      case '&': return '&amp;';
      case '<': return '&lt;';
      case '>': return '&gt;';
      case '"': return '&quot;';
      case "'": return '&#039;';
      default: return m;
    }
  });
}
```

- **Purpose**: Safely escapes HTML special characters to prevent XSS attacks.

2. Escape RegExp Special Characters

```
function escapeRegExp(string) {
  return string.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
}
```

- **Purpose**: Escapes special characters in a string to safely use it in a regular expression.

3. Get File Extension

```
function getFileExtension(filePath) {
  return path.extname(filePath).toLowerCase();
}
```

- **Purpose**: Extracts the file extension from a given file path.

4. Get File Name Without Extension

```
function getFileNameWithoutExtension(filePath) {
  return path.basename(filePath, path.extname(filePath));
}
```

- **Purpose**: Extracts the file name minus its extension.

5. Inclusion Check Based on File Extension

```
function shouldIncludeFile(filePath, allowedExtensions) {
  const ext = getFileExtension(filePath);
  return allowedExtensions.includes(ext);
}
```

- **Purpose**: Determines whether a file should be included based on a list of allowed extensions.

6. Format Bytes to Human Readable Format

```
function formatBytes(bytes, decimals = 2) {
  if (bytes === 0) return '0 Bytes';

  const k = 1024;
  const dm = decimals < 0 ? 0 : decimals;
  const sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB'];

  const i = Math.floor(Math.log(bytes) / Math.log(k));

  return parseFloat((bytes / Math.pow(k, i)).toFixed(dm)) + ' ' + sizes[i];
}
```

- **Purpose**: Converts bytes into a more readable format with the appropriate size suffix.

7. Capitalize the First Letter of a String

```
function capitalizeFirstLetter(string) {
  return string.charAt(0).toUpperCase() + string.slice(1);
}
```

- **Purpose**: Capitalizes the first letter of a given string.

8. Convert camelCase to kebab-case

```
function camelToKebab(string) {
  return string.replace(/([a-z0-9])([A-Z])/g, '$1-$2').toLowerCase();
}
```

- **Purpose**: Converts camelCase strings to kebab-case.

9. Truncate a String with Ellipsis

```
function truncateString(str, maxLength) {
  if (str.length <= maxLength) return str;
  if (maxLength <= 3) return '...';
  return str.slice(0, maxLength - 3) + '...';
}
```

- **Purpose**: Truncates a string to a specified maximum length, adding ellipsis if necessary.

10. Generate a Unique ID

```
function generateUniqueId() {
  return crypto.randomBytes(8).toString('hex');
}
```

- **Purpose**: Generates a unique 16-character hexadecimal ID.

11. Sleep for a Specified Duration

```
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

- **Purpose**: Pauses execution for a specified number of milliseconds, useful for asynchronous operations.

12. Deep Merge Two Objects

```
function deepMerge(target, source) {
  if (typeof target !== 'object' || target === null) {
    return source;
  }

  if (typeof source !== 'object' || source === null) {
    return target;
  }

  const output = Array.isArray(target) ? [...target] : { ...target };

  for (const key in source) {
    if (source.hasOwnProperty(key)) {
      if (isObject(source[key])) {
        output[key] = deepMerge(output[key] || {}, source[key]);
      } else {
        output[key] = source[key];
      }
    }
  }

  return output;
}
```

- **Purpose**: Merges two objects recursively, useful for copying nested objects.

13. Check if a Value is an Object

```
function isObject(item) {
  return (item && typeof item === 'object' && !Array.isArray(item));
}
```

- **Purpose**: Checks whether the given value is a non-null object.

14. Format Date to Readable String

```
function formatDate(date) {
  return date.toLocaleDateString('en-US', {
    year: 'numeric',
    month: 'long',
    day: 'numeric',
    hour: '2-digit',
    minute: '2-digit'
  });
}
```

- **Purpose**: Converts a JavaScript `Date` object to a readable string format.

15. Pluralize a Word Based on Count

```
function pluralize(count, singular, plural) {
  return count === 1 ? singular : plural;
}
```

- **Purpose**: Chooses the correct singular or plural form of a word based on a given count.

16. Convert Array to Human-Readable List

```
function arrayToHumanReadableList(array) {
  if (array.length === 0) return '';
  if (array.length === 1) return array[0];
  if (array.length === 2) return array.join(' and ');

  const arrCopy = [...array];
  const last = arrCopy.pop();
  return arrCopy.join(', ') + ', and ' + last;
}
```

- **Purpose**: Converts an array into a grammatically correct list string.

17. Dump Data and Terminate Process

```
function dd(data) {
  process.stdout.write(JSON.stringify(data, null, 2) + '\n');
  process.exit();
}
```

- **Purpose**: Outputs data to the console in a readable JSON format and then terminates the script's execution.

Export

Finally, all these utility functions are exported for use in other modules:

```
module.exports = {
  dd,
  escapeHtml,
  escapeRegExp,
  getFileExtension,
  getFileNameWithoutExtension,
  shouldIncludeFile,
  formatBytes,
  capitalizeFirstLetter,
  camelToKebab,
  truncateString,
  generateUniqueId,
  sleep,
  deepMerge,
  isObject,
  formatDate,
  pluralize,
  arrayToHumanReadableList
};
```

These functions can be imported and used in other parts of your application, making it easier to perform common operations in a consistent and efficient manner.

```
This explanation provides a clear understanding of each function's purpose and usage within the `utils.js` file.
```