```
/**
* Collection de mÃ©thodes utilitaires JavaScript
*/
class UtilityMethods {
 /**
  * Formate un nombre en prix avec devise
  * @param {number} amount - Le montant Ã  formater
  * @param {string} currency - Le code de la devise (par dÃ©faut: EUR)
  * @param {string} locale - La locale Ã  utiliser (par dÃ©faut: fr-FR)
  * @returns {string} Le prix formatÃ©
  */
 formatPrice(amount, currency = "EUR", locale = "fr-FR") {
  return new Intl.NumberFormat(locale, {
   style: "currency",
   currency: currency,
  }).format(amount);
 }

 /**
  * GÃ©nÃ¨re un identifiant unique
  * @param {number} length - La longueur souhaitÃ©e de l'identifiant (par dÃ©faut: 10)
  * @returns {string} L'identifiant unique gÃ©nÃ©rÃ©
  */
 generateUniqueId(length = 10) {
  const characters =
   "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
  let result = "";
  for (let i = 0; i < length; i++) {
```

```
    result += characters.charAt(
      Math.floor(Math.random() * characters.length)
    );
  }
  return result;
}

/**
 * Tronque un texte Ã  une longueur maximale
 * @param {string} text - Le texte Ã  tronquer
 * @param {number} maxLength - La longueur maximale souhaitÃ©e
 * @param {string} suffix - Le suffixe Ã  ajouter (par dÃ©faut: "...")
 * @returns {string} Le texte tronquÃ©
 */
truncateText(text, maxLength, suffix = "...") {
  if (text.length <= maxLength) return text;
  return text.substring(0, maxLength - suffix.length) + suffix;
}

/**
 * Valide une adresse email
 * @param {string} email - L'adresse email Ã  valider
 * @returns {boolean} True si l'email est valide, false sinon
 */
validateEmail(email) {
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return emailRegex.test(email);
}
```

```
/**
 * Calcule le temps Ã©coulÃ© depuis une date donnÃ©e
 * @param {Date} date - La date de rÃ©fÃ©rence
 * @returns {string} Le temps Ã©coulÃ© en format lisible
 */
getTimeElapsed(date) {
  const seconds = Math.floor((new Date() - date) / 1000);

  const intervals = {
    annÃ©e: 31536000,
    mois: 2592000,
    semaine: 604800,
    jour: 86400,
    heure: 3600,
    minute: 60,
  };

  for (const [unit, secondsInUnit] of Object.entries(intervals)) {
    const interval = Math.floor(seconds / secondsInUnit);
    if (interval >= 1) {
      return ` Il y a ${interval} ${unit}${interval > 1 ? "s" : ""}` ;
    }
  }

  return "Ã€ l'instant";
}
```

```
/**
 * MÃ©lange alÃ©atoirement les Ã©lÃ©ments d'un tableau
 * @param {Array} array - Le tableau Ã  mÃ©langer
 * @returns {Array} Le tableau mÃ©langÃ©
 */
shuffleArray(array) {
  const shuffled = [...array];
  for (let i = shuffled.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [shuffled[i], shuffled[j]] = [shuffled[j], shuffled[i]];
  }
  return shuffled;
}

/**
 * Convertit une chaÃ®ne en slug URL
 * @param {string} string - La chaÃ®ne Ã  convertir
 * @returns {string} Le slug gÃ©nÃ©rÃ©
 */
generateSlug(string) {
  return string
    .toLowerCase()
    .normalize("NFD")
    .replace(/[\u0300-\u036f]/g, "")
    .replace(/[^a-z0-9\s-]/g, "")
    .replace(/\s+/g, "-")
    .replace(/-+/g, "-")
    .trim();
```

```
}

/**
 * VÃ©rifie si un objet est vide
 * @param {Object} obj - L'objet Ã  vÃ©rifier
 * @returns {boolean} True si l'objet est vide, false sinon
 */
isEmptyObject(obj) {
  for (const prop in obj) {
    if (Object.hasOwn(obj, prop)) return false;
  }
  return true;
}

/**
 * Groupe les Ã©lÃ©ments d'un tableau par une propriÃ©tÃ©
 * @param {Array} array - Le tableau Ã  grouper
 * @param {string} key - La propriÃ©tÃ© Ã  utiliser pour le groupement
 * @returns {Object} L'objet groupÃ©
 */
groupBy(array, key) {
  return array.reduce((acc, current) => {
    const groupKey = current[key];
    if (!acc[groupKey]) {
      acc[groupKey] = [];
    }
    acc[groupKey].push(current);
    return acc;
```

```
  }, {});
}

/**
 * Effectue une requÃªte HTTP avec gestion du dÃ©lai d'attente
 * @param {string} url - L'URL de la requÃªte
 * @param {Object} options - Les options de la requÃªte
 * @param {number} timeout - Le dÃ©lai d'attente en millisecondes
 * @returns {Promise} La promesse de la requÃªte
 */
async fetchWithTimeout(url, options = {}, timeout = 5000) {
  const controller = new AbortController();
  const id = setTimeout(() => controller.abort(), timeout);

  try {
    const response = await fetch(url, {
      ...options,
      signal: controller.signal,
    });
    clearTimeout(id);
    return response;
  } catch (error) {
    clearTimeout(id);
    if (error.name === "AbortError") {
      throw new Error(
        "La requÃªte a Ã©tÃ© abandonnÃ©e : dÃ©lai d'attente dÃ©passÃ©"
      );
    }
```

```
    throw error;
  }
 }
}

// Exemple d'utilisation
const utils = new UtilityMethods();

// Exemples d'utilisation des mÃ©thodes
console.log(utils.formatPrice(42.99)); // 42,99 â,¬
console.log(utils.generateUniqueId()); // ChaÃ®ne alÃ©atoire de 10 caractÃ¨res
console.log(utils.truncateText("Un trÃ¨s long texte", 10)); // "Un trÃ¨s..."
console.log(utils.validateEmail("test@example.com")); // true
console.log(utils.getTimeElapsed(new Date("2024-01-01"))); // "Il y a X mois"
console.log(utils.shuffleArray([1, 2, 3, 4, 5])); // Array mÃ©langÃ©
console.log(utils.generateSlug("Voici un Titre d'Article!")); // "voici-un-titre-d-article"
console.log(utils.isEmptyObject({})); // true
console.log(
  utils.groupBy([{ type: "A" }, { type: "B" }, { type: "A" }], "type")
); // GroupÃ© par type
```