Découvrez les sources de bug potentielle

Concept	Signification et Exemples
Bugs Front-End	Les défauts dans la partie front-end d'un site web, tels que des erreurs d'affichage ou JavaScript.
Exemples de Bugs	1. Problèmes CSS sur smartphones. 2. Erreurs JavaScript dans la console.
Sources de Bugs	1. Côté HTML: Balises non fermées, mauvaise utilisation des classes. 2. Côté CSS: Affichage incorrect sur certains appareils. 3. Côté DOM: Sélection, modification ou suppression incorrecte des nœuds. 4. Côté API: Mauvaise utilisation d'API interne ou externe.
Cascade de Bugs	Propagation des erreurs, p. ex., une défaillance d'API influençant la création d'éléments.

Prenez en main les outils de debogage

Concept	Signification et Exemples
Console	Un environnement de programmation interactif permettant d'initialiser des variables, analyser les erreurs JavaScript, afficher des informations, et écrire des fonctions directement dans le navigateur.
Exemple Console	1. Initialiser une variable : `let x = 5;` 2. . Afficher une information : `console.log("Hello, World!");`
Inspecteur	Un outil pour déboguer le CSS et le HTML, visualiser les règles d'espacement du modèle de boîte, et observer le comportement de l'interface sur différents appareils et résolutions.
Exemple Inspecteur	1. Modifier les règles CSS directement dans l'inspecteur. 2. Observer le modèle de boîte pour résoudre des bugs visuels.
Débogueur	Un outil permettant de suivre l'état du code étape par étape, en ajoutant des points de rupture pour comprendre le comportement de l'application de manière détaillée.
Exemple Débogueur	1. Ajouter un point de rupture pour observer l'état des variables à une certaine étape du code.
Extension VsCode	Une extension pour l'éditeur de texte VsCode qui facilite le débogage en permettant d'utiliser le débogueur avec Google Chrome.
Onglet Network	Un outil permettant de traquer toutes les requêtes et réponses du navigateur, essentiel pour déboguer les requêtes HTTP et connaître le poids des fichiers reçus.
Exemple Onglet Network	1. Suivre les requêtes HTTP pour déboguer les appels réseau. 2. Connaître le poids des fichiers reçus pour optimiser le chargement du site.

Adoptez logique de debogage

Concept	Signification et Exemples
Méthodologie de Débogage	Un processus en plusieurs étapes pour résoudre les bugs de manière efficace. Les étapes comprennent l'observation du bug, l'écriture d'un test répétable, l'élaboration et le test de théories, et enfin la résolution du bug.
Étape 1: Observer le Bug	Prendre du recul, observer la situation, et poser des questions sur le bug. Exemples de questions : Où se trouve le bug ? Y a-t-il une erreur dans la console ? Quel type de bug est-ce à première vue ?
Étape 2: Test Répétable	Reproduire le bug dans votre environnement de développement. Si le bug se produit sur l'ordinateur d'un utilisateur, collecter des informations sur l'appareil, le système d'exploitation, et le navigateur. Écrire du code pour reproduire les conditions du bug.
Étape 3: Élaborer et Tester	Formuler des hypothèses sur l'origine du bug, basées sur les observations précédentes. Tester ces hypothèses en modifiant le code. Si la théorie échoue, ajuster et réessayer. La règle du Boy Scout s'applique : corriger tous les bugs identifiés.
Étape 4: Résoudre le Bug	Une fois que la théorie est confirmée et que le bug est résolu, s'assurer que le code reste propre et corriger tous les bugs identifiés au cours du processus de débogage.

Détectez bug HTML et CSS

Concept	Signification et Exemples
Bugs d'intégration	Problèmes rencontrés du côté HTML et CSS lors de l'intégration. Souvent difficiles à détecter car ils ne génèrent pas d'erreurs dans la console. Les causes incluent des erreurs HTML, des problèmes de CSS, et des incompatibilités entre navigateurs.
Erreurs liées au HTML	Souvent causées par l'oubli de fermeture d'une balise ou l'utilisation de balises dépréciées. Exemple : une balise non fermée peut entraîner un affichage incorrect, comme des blocs mal positionnés sur la page.
Erreurs liées au comportement CSS	Résultent souvent du non-respect des règles de cascading et de spécificités CSS. Les bugs peuvent survenir lorsque les propriétés CSS sont surchargées. Exemple : une propriété CSS surchargée peut causer un comportement inattendu du texte.
Erreurs liées aux navigateurs	Liées à des incompatibilités entre navigateurs. Par exemple, des bugs peuvent survenir si un navigateur ne prend pas en charge une fonctionnalité CSS spécifique. Exemple : l'utilisation de "Vendor Prefixes" pour assurer la compatibilité entre navigateurs.
Utilisation de l'inspecteur	Outil de développement permettant d'analyser le code HTML et CSS en changeant la résolution, le type de dispositif, inspectant les modèles de boîte, et vérifiant les règles CSS surchargées ou non appliquées.
Utilisation d'un validateur HTML/CSS	Recours à un validateur pour détecter des erreurs ou avertissements dans le code. Exemple : un validateur HTML peut signaler une balise mal fermée, tandis qu'un validateur CSS peut détecter des erreurs de syntaxe.
Utilisation de CanIUse	Site web permettant d'analyser la compatibilité des fonctionnalités CSS ou JavaScript avec différents navigateurs. Utile pour comprendre les avertissements des validateurs et assurer une meilleure compatibilité entre navigateurs.
Linters	Outils intégrés aux éditeurs de texte pour détecter des erreurs de style et des oublis dans le code. Exemple : EsLint est un linter populaire en développement front-end qui peut signaler des problèmes de code JavaScript.

Tirez parti de l’inspecteur pour résoudre vos bug

Concept	Signification et Exemples
Inspectez votre HTML	Utilisation de l'inspecteur pour visualiser et éditer le code HTML en direct. Permet de tester rapidement des hypothèses de débogage en modifiant des éléments, par exemple en ajoutant ou supprimant une classe. Les modifications ne sont pas sauvegardées après le rafraîchissement de la page.
Inspectez votre CSS	Utilisation de l'inspecteur pour examiner et modifier les règles CSS en direct. Permet de voir la ligne précise où une règle CSS est définie. Utile pour déboguer des problèmes CSS, tels que des bugs d'espacement ou des règles surchargées. Dans un environnement de développement, le CSS peut être écrit en ligne, tandis qu'en production, il est optimisé et regroupé.
Environnement de développement et de production	Distinction entre l'environnement de développement, conçu pour fournir des informations détaillées aux développeurs, et l'environnement de production, optimisé pour la performance. L'inspecteur CSS permet d'observer le comportement du modèle de boîte, ce qui est utile pour résoudre des bugs d'espacement.
Bugs d'intégration	Problèmes rencontrés dans l'intégration du HTML et du CSS. Les erreurs d'intégration peuvent inclure des oublis de fermeture de balises, des balises dépréciées, des erreurs de sélection CSS, et des incompatibilités entre navigateurs. L'utilisation de l'inspecteur permet de détecter et résoudre ces problèmes.
Modèle de boîte	Concept CSS décrivant comment les éléments HTML sont rendus et comment l'espace est distribué autour d'eux. L'inspecteur CSS permet d'observer le modèle de boîte, ce qui est particulièrement utile pour résoudre des bugs d'espacement
Cascade CSS	Processus par lequel les règles CSS sont appliquées aux éléments, en fonction de leur spécificité et de l'ordre de déclaration. L'inspecteur CSS aide à détecter les règles surchargées et à comprendre les problèmes de cascade CSS pouvant causer des bugs d'intégration.
Bugs d'espacement	Problèmes liés à la mise en page et à la répartition de l'espace entre les éléments HTML. L'inspecteur CSS est utile pour identifier et résoudre ces bugs en examinant le modèle de boîte et en ajustant les propriétés telles que margin et padding.
Débogage CSS	Utilisation de l'inspecteur pour déboguer des problèmes CSS, y compris la modification de règles en direct, la détection de règles surchargées, et la compréhension des problèmes de cascade CSS.
Débogage HTML	Utilisation de l'inspecteur pour déboguer des problèmes HTML en visualisant et éditant le code en direct. Permet de tester des hypothèses rapidement et d'identifier des erreurs telles que des balises mal fermées.

Tirez partie de l’inspec pour résoudre bugs

Concept	Signification et Exemples
Inspectez votre HTML	Utilisation de l'inspecteur pour visualiser et éditer le code HTML en direct. Permet de tester rapidement des hypothèses de débogage en modifiant des éléments, par exemple en ajoutant ou supprimant une classe. Les modifications ne sont pas sauvegardées après le rafraîchissement de la page.
Inspectez votre CSS	Utilisation de l'inspecteur pour examiner et modifier les règles CSS en direct. Permet de voir la ligne précise où une règle CSS est définie. Utile pour déboguer des problèmes CSS, tels que des bugs d'espacement ou des règles surchargées. Dans un environnement de développement, le CSS peut être écrit en ligne, tandis qu'en production, il est optimisé et regroupé.
Environnement de développement et de production	Distinction entre l'environnement de développement, conçu pour fournir des informations détaillées aux développeurs, et l'environnement de production, optimisé pour la performance. L'inspecteur CSS permet d'observer le comportement du modèle de boîte, ce qui est utile pour résoudre des bugs d'espacement.
Modèle de boîte	Concept CSS décrivant comment les éléments HTML sont rendus et comment l'espace est distribué autour d'eux. L'inspecteur CSS permet d'observer le modèle de boîte, ce qui est particulièrement utile pour résoudre des bugs d'espacement.

Bugs d'intégration	Problèmes rencontrés dans l'intégration du HTML et du CSS. Les erreurs d'intégration peuvent inclure des oublis de fermeture de balises, des balises dépréciées, des erreurs de sélection CSS, et des incompatibilités entre navigateurs. L'utilisation de l'inspecteur permet de détecter et résoudre ces problèmes.
Cascade CSS	Processus par lequel les règles CSS sont appliquées aux éléments, en fonction de leur spécificité et de l'ordre de déclaration. L'inspecteur CSS aide à détecter les règles surchargées et à comprendre les problèmes de cascade CSS pouvant causer des bugs d'intégration.
Bugs d'espacement	Problèmes liés à la mise en page et à la répartition de l'espace entre les éléments HTML. L'inspecteur CSS est utile pour identifier et résoudre ces bugs en examinant le modèle de boîte et en ajustant les propriétés telles que margin et padding.
Débogage CSS	Utilisation de l'inspecteur pour déboguer des problèmes CSS, y compris la modification de règles en direct, la détection de règles surchargées, et la compréhension des problèmes de cascade CSS.
Débogage HTML	Utilisation de l'inspecteur pour déboguer des problèmes HTML en visualisant et éditant le code en direct. Permet de tester des hypothèses rapidement et d'identifier des erreurs telles que des balises mal fermées.

Découvrez les différents bug du DOM

Concept	Signification et Exemples
Bugs liés à la sélection de nœuds	Découverte des erreurs fréquentes liées à la sélection d'éléments via le DOM ou des API, incluant les erreurs de référence et les erreurs silencieuses. Importance de comprendre le type des objets manipulés et des méthodes telles que `querySelector` et `querySelectorAll`.
Bugs de création de nœuds	Exploration des problèmes courants liés à la création dynamique de nouveaux éléments dans le DOM. Mise en garde contre les erreurs de sélection d'éléments inexistants et l'utilisation de méthodes comme `innerHTML` ou `insertAdjacentHTML`, préconisant plutôt une approche séparée avec `createElement`, `textContent`, `classList`, et `appendChild`.
Bugs de modification de nœuds existants	Identification des erreurs fréquentes lors de la tentative d'utilisation de propriétés ou méthodes inexistantes sur des nœuds sélectionnés. Nécessité de comprendre le type des variables manipulées et d'interpréter les messages d'erreur associés.
Utilisation du CSS pour le style	Discussion sur l'ajout de style aux éléments HTML via JavaScript. Comparaison entre l'ajout/suppression de classes pour appliquer le style et l'ajout direct de style via la propriété `.style`. Préférence pour l'utilisation des classes pour respecter les principes de responsabilité unique, distinguant le HTML (contenu), le CSS (mise en page), et le JavaScript (interactions).
Bugs liés à la suppression de nœuds	Exploration des erreurs courantes liées à la suppression de nœuds du DOM. Mise en garde contre la confusion entre masquer et supprimer un nœud, ainsi que l'utilisation incorrecte de la méthode `removeChild()`. Présentation d'une erreur potentielle lorsqu'on ne sélectionne pas directement le nœud parent avant d'utiliser `removeChild()`.

Servez vous des outils lié au debogage

Concept	Signification et Exemples
Utilisation de la console pour déboguer	Ce cours explore l'utilisation avancée de la console et du débogueur pour résoudre les problèmes de code. La console est un outil polyvalent permettant d'initialiser et d'accéder à des variables, d'analyser des erreurs JavaScript, d'afficher et de filtrer des informations, ainsi que d'écrire et d'exécuter des fonctions directement dans l'environnement de développement. L'auto-complétion, la gestion des erreurs, l'affichage sélectif d'informations et l'exécution de fonctions offrent des fonctionnalités puissantes pour le débogage efficace.
Initialiser et accéder à des variables	La console permet d'initialiser de nouvelles variables directement dans l'environnement de développement et de récupérer des variables existantes déclarées dans le code source. Cette fonctionnalité est utile pour tester rapidement du code, comprendre l'état d'une variable pendant l'exécution du programme et faciliter le débogage en affichant des valeurs en temps réel.
Analyser des erreurs JavaScript	La console est utilisée pour analyser les erreurs JavaScript, fournissant des messages d'erreur détaillés. Comprendre ces messages est crucial pour localiser et résoudre les erreurs dans le code. L'exemple montre une erreur de référence où la variable appelée n'existe pas, soulignant l'importance de comprendre les types d'erreurs et leur contexte.
Afficher et filtrer des informations	En plus de l'affichage d'informations avec `console.log`, la console permet de filtrer les niveaux d'information, par exemple, en activant la réception des avertissements CSS uniquement. Cela permet de cibler spécifiquement les types d'informations nécessaires pour le débogage, facilitant l'identification des problèmes et la compréhension du flux d'exécution du code.
Écrire puis exécuter des fonctions	La console offre la possibilité d'écrire et d'exécuter des fonctions directement. Cette fonctionnalité est pratique pour tester des portions de code, évaluer des expressions et vérifier le comportement des fonctions sans avoir à les intégrer dans le code source principal. Elle offre une approche interactive pour expérimenter avec le code et valider des concepts rapidement.

Récupérez des données sur une API externe

Concept	Signification et Exemples
Développement avec les API	Les développeurs front-end utilisent quotidiennement les API pour récupérer des données (météo, séances de cinéma) et interagir avec des fonctionnalités du navigateur. Cette partie aborde le débogage des API externes (requêtes à des services externes) et internes (intégrées au navigateur), avec un accent sur la récupération de données météo via l'API WeatherStack.
Récupération des données météo	Le projet Façadia illustre la récupération de données météo via l'API WeatherStack. Deux approches sont utilisées : un fichier de données mockées (weather-api-mocked-data.json) pour les tests et une requête directe à l'API. La page dédiée aux capteurs comprend deux tableaux, l'un pour les données du capteur et l'autre pour les données météo.
Découverte de l'API WeatherStack	WeatherStack, une API météo complète et gratuite, est présentée. Avant d'intégrer l'API au code JavaScript, il est recommandé de la découvrir avec Postman, parcourir la documentation, et comprendre les réponses et le format des données. Cette étape préliminaire facilite le développement en évitant des heures de débogage JavaScript en cas de problème avec l'API.
Intégration de l'API WeatherStack	L'intégration de l'API WeatherStack côté JavaScript utilise la méthode fetch et les promesses. Deux approches sont présentées : l'utilisation d'un fichier de données mockées (weather-api-mocked-data.json) pour les tests et une fonction dédiée pour les requêtes réelles à l'API en fonction des données géographiques du capteur. Les requêtes et leur fonctionnement sont détaillés dans une vidéo associée.
Compréhension des messages d'erreur d'une API	Pour bien déboguer une API, il est crucial de comprendre les codes HTTP renvoyés par l'API et les messages d'erreur. Les bonnes pratiques incluent l'utilisation de noms plutôt que de verbes pour les endpoints, les status codes HTTP pour traiter les erreurs proprement, et l'inclusion de la version de l'API dans l'URL.
Découverte des messages d'erreur d'une API	La gestion des messages d'erreur d'une API est explorée. Les développeurs doivent connaître les codes HTTP et les messages en cas d'erreurs. Les bonnes pratiques pour les API publiques incluent l'utilisation de noms au lieu de verbes pour les endpoints et la présence de la version de l'API dans l'URL. Des exemples de messages d'erreur de l'API WeatherStack sont analysés.

Récupérez données sur une API interne

Concept	Signification et Exemples
API dans le navigateur	Les navigateurs offrent de nombreuses API internes, telles que le Web Storage avec LocalStorage et l'API Canvas pour dessiner avec JavaScript et HTML. Ces API facilitent l'interaction des développeurs avec le navigateur et offrent diverses fonctionnalités. Certains exemples sont le stockage d'informations avec LocalStorage et le dessin via l'API Canvas. Le MDN fournit une liste exhaustive des API disponibles.
Récupération de données de géolocalisation	Dans le projet Façadia, l'API de géolocalisation est utilisée sur la page de création d'un formulaire. La documentation sur le MDN offre des informations claires sur son utilisation. La méthode getCurrentPosition permet de récupérer la position actuelle, et watchPosition permet de suivre l'évolution de la position. Dans ce contexte, mocker les données n'est pas nécessaire en raison de l'absence de limite de consommation de l'API de géolocalisation.
Intégration de l'API de géolocalisation	L'API de géolocalisation est intégrée dans le projet Façadia sur la page de création d'un capteur. L'utilisation du code source de la branche principale est illustrée, montrant comment l'API est intégrée au projet. La récupération de données de géolocalisation est présentée comme un complément à l'intégration de l'API dans le projet.
Compréhension des messages d'erreur d'une API	La gestion des erreurs des API internes peut varier, mais la documentation du MDN est généralement excellente. Les erreurs peuvent différer d'une API à l'autre, mais la documentation fournit des informations détaillées sur la façon de traiter les erreurs de manière élégante. L'exemple de l'API de géolocalisation montre comment gérer les erreurs en passant des callbacks dans la méthode getCurrentPosition. Le traitement des erreurs est considéré comme obligatoire pour éviter des problèmes graves de programme.
Découverte de la gestion des erreurs de géolocalisation	La documentation du MDN sur l'API de géolocalisation explique comment gérer les erreurs, en particulier dans le contexte de la méthode getCurrentPosition. Elle souligne l'importance de traiter les cas d'erreur en passant des callbacks de succès et d'erreur. L'exemple de code montre comment mettre en œuvre ces callbacks pour réagir de manière appropriée aux situations où la récupération de la position échoue. Le traitement des erreurs est considéré comme une étape essentielle pour éviter les problèmes et assurer une expérience utilisateur fluide.

Découvrez un bug complexe

Concept	Signification et Exemples
Contextualisation d'un bug	Lors du débogage, les bugs peuvent être isolés ou interconnectés. Si un bug affecte uniquement une page spécifique, les méthodes de débogage standard peuvent être appliquées. Cependant, dans des applications complexes, un bug peut déclencher d'autres bugs, créant une cascade d'erreurs. Il est crucial de reconnaître que l'erreur visible peut être la partie émergée de l'iceberg et que le bug peut également résider du côté du serveur, de l'API, ou des données.
Identification de l'origine d'un bug	Les bugs peuvent avoir des origines variées, y compris le code côté client, le serveur, l'API, ou même la base de données. Une analyse attentive du code peut révéler la source du problème. Par exemple, des modifications non prévues dans une application Python ont provoqué des erreurs, soulignant l'importance de l'analyse du code pour détecter les problèmes. Les bugs peuvent également provenir de traitements asynchrones complexes, comme l'expérience d'un développeur avec Firebase.
Méthodologie de débogage	La méthodologie de débogage implique plusieurs étapes clés : observer les bugs, écrire un test répétable, élaborer et tester des théories, puis résoudre le bug. Ces étapes fournissent une structure pour aborder les problèmes et sont essentielles, surtout pour les débutants. La réflexion sur ces étapes et l'utilisation instinctive de la méthodologie deviennent naturelles avec l'expérience.
Isolation d'un bug majeur	Face à des bugs complexes, il est essentiel de ne pas essayer de déboguer tout en une fois. L'histoire d'un problème d'envoi d'e-mails montre l'importance d'isoler le bug. En fournissant des informations telles que les messages d'erreur, le code utilisé, et les URL pertinentes, il devient possible de cibler la cause. Dans l'exemple, l'isolation a conduit à la découverte d'une configuration incorrecte du serveur Smtp, résolvant ainsi le problème.
Apprentissage de l'isolation des bugs majeurs	L'histoire de l'envoi d'e-mails souligne l'importance d'isoler un bug. En se concentrant sur des aspects spécifiques, tels que la bibliothèque utilisée (PhpMailer), un script PHP de test a pu être créé pour isoler le problème. Le remplacement des informations de connexion a ensuite révélé que le problème résidait dans la configuration du serveur Smtp. Apprendre à isoler les bugs majeurs est crucial pour éviter de perdre du temps à déboguer des aspects non pertinents du système.

Résolvez un bug complexe

Concept	Signification et Exemples
Utilisation des outils de débogage	Pour résoudre efficacement les bugs, plusieurs outils de débogage sont disponibles. L'inspecteur est idéal pour examiner le HTML et le CSS, révélant les propriétés CSS utilisées et le modèle de boîte. La console est essentielle pour le débogage du code JavaScript, permettant l'exécution de fonctions et l'utilisation de `console.log` pour suivre l'état de l'application. Le débogueur, avec des breakpoints, offre une visibilité approfondie dans le JavaScript. L'onglet Network révèle les requêtes HTTP et les réponses.
Réflexion sur l'implémentation	Après avoir isolé un bug et codé une solution, la réflexion sur l'implémentation est cruciale. Il est nécessaire de se demander si la solution couvre d'autres cas d'erreur et si elle pourrait entraîner d'autres bugs. Cette étape vise à prévenir les erreurs futures et à assurer une mise en œuvre robuste. "Faire le tour du propriétaire" avant de passer à autre chose évite les erreurs inattendues et les impacts non prévus.
Prévention des erreurs futures	Même après la résolution d'un bug, il est essentiel de prévenir les erreurs futures. Des questions telles que la couverture des cas d'erreur par la solution, la possibilité de nouveaux bugs, et les mesures pour éviter que l'erreur ne se reproduise sont cruciales. Cette approche proactive vise à maintenir la stabilité du code et à éviter les régressions lors de mises à jour ou de modifications du projet.
Tests automatisés et monitoring	Dans des projets d'envergure, il est nécessaire d'aller au-delà de l'inspection manuelle. Les tests automatisés et les outils de monitoring sont des étapes avancées dans le processus de développement. Les tests automatisés assurent une vérification continue du bon fonctionnement de l'application, tandis que le monitoring permet de détecter les problèmes en temps réel, offrant une approche proactive pour assurer la qualité du logiciel.