ZCOURS

SPRING

Spring Core container :
+ Pattern IoC (Inversion de contrôle) : fournit conteneur / gère et met en œuvre composants (beans) / Applique configuration / Fournit contexte applicatif
+ ApplicationContext : conteneur lié au context applicatif
+ Composants : beans sont des POJOS / instanciés par le conteneur (nom, classe, dépendances, scope) / Injection de dépendances / Scopes (request, session, prototype…)
+ Configuration : par fichiers XML / par annotations (@Component, @Controller…)
Spring Web MVC :
+ MVC de type 2 / + WebApplicationContext : root (1 contexte commun de l'app) / servlet (contextes spécifiques à chaque DispatcherServlet) / Configuration obligatoire (web.xml)
+ Composants MVC : @Controller / @RequestMapping
+ Spring Web MVC (framework) : méthodes de service (@ModelAttribute)
Spring Boot (framework) :
+ Puissant mais lourd à configurer / simplifier la configuration des projets / convention over configuration
+ Créer facilement une application Java standalone / pas de config XML ni génération de code ni fichier
+ Projet Maven / plugins et dépendances dans pom.xml
+ Fonctionnement : @EnableAutoConfiguration / méthode main
Avantages :
+ Légèreté (lol) / s'appuie sur des solutions open sources éprouvées / possibilité de « plugger » d'autres fonctionnalités / configruation rapide des applications « simples » / très utilisé / documentation abondante
+ Faiblesse : complexité croissante (beaucoup de sous-projets / 3 types de configurations possibles /

MASHUP DE DONNÉES CÔTÉ CLIENT

+ Origin : identifie l'endroit du Web d'où provient une ressource
+ Same-Origin Policy : Permet scripts d'une page d'accéder sans restriction à tous objets de la page, toutes méthodes des autres scripts, tous objets des autres pages de même origine / Interdit accès à ces éléments depuis les pages d'origines différentes / Interdit envoi de requêtes vers origines différentes de celle de la ressource contenant le script / exceptions : images, scripts, formulaires / but : éviter pertes – modif de données, vols de session, perte de confidentialité
+ Mashup : utilisation ressources issues de plusieurs origines / récupération données à partir sources externes (Linked Data, Web APIs, capteurs physiques)
+ Web APIs : services accessibles pour permettre de fournir – transformer des données / certaines fournissent interfaces RESTful
+ Cross-Origin : permettre envoi et traitement de requêtes à d'autres serveurs que celui d'origine d'une page / initiatives : proxy serveur, JSONP (JSON with Padding), CORS
+ JSONP : exploiter exceptions à la same-origin policy pour envoyer des requêtes cross-domain
+ CORS : remplacer la same-origin police par un mécanisme d'autorisation plus fin / fonctionnement : ensemble de headers http permettant de contrôler l'utilisation de ressources cross-origin (headers requêtes et réponses et autres) / client ajoute header Origin à requête pour indiquer origine script qui l'envoie / serveur renvoie header Access-Control-Allow-Origin dans réponse pour indiquer qu'il accepte
+ CORS with preflight : client envoie une requête avec méthode « OPTIONS » pour demander au serveur s'il peut utiliser methode http particulière, un type de header particuler / serveur répond en indiquant les méthodes http qu'il accepte, s'il autorise l'identification de l'utilisateur / interdiction de passer de HTTPS à HTTP

OUTIL DE PROGRAMMATION EN JS

Node.js :
+ plateforme applicative pour JS / pas de BOM, paquets spécifiques / s'appuie sur spec CommonJS (format de def de modules, bibli d'import async require.js) / fondé sur notion event loop (exec callbacks en async) / s'appuie sur mécanismes multitasking de l'OS
+ gestion des dépendances : notion de module, intallation avec NPM, propre commande require (fonctionne pas côté client)
NPM :
+ gestionnaire de paquets pour node / registery de paquets / CLI comme apt-get / description dans package.json
+ npm init / npm install / npm install <module>@1.0.1 / npm install -g <package>
BOWER : gestionnaire de paquets applicatif côté client / descriptif dans bower.json / diff NPM : optimisé front-end
GRUNT : task manager – runner / ensemble de plugins / config projet : Gruntfile / utilisation via CLI
Webpack : gestion de module (require/import) / agnostique au type de module, traite assets statiques comme des modules (css, images) / découpe code pour optimiser chargement / injection de dépendance et multi-compilation / s'intègre aux autres outils de build
Autres : Express (node), Test : Mocha ou Jasmine, Verification : ESLint, Compilation : Babel / alternative NPM : Yarn

FRAMEWORK CÔTÉ CLIENT

+ Objectif : faciliter dev app « single-page » côté client / gérer logique métier / s'adapter terminaux mobiles / interagir avec serveurs / optimiser perf
+ Propriétés : pattern MV* / modularité / réactivité
Propriétés :
+ Routage : simuler pages web différentes (changement hash URL, récupérer paramètres, déclencher callback)
+ Liens entre modèle et vue : one-way data binding (toute modif d'une prop du modèle provoque une maj de la vue) / two-way data binding (owdb + action sur la vue provoque maj du modèle)
+ Templating : côté client (produire vue à partir de données, ex : Mustache, Handlebars) / interpolations / directives (if)
+ Composants : étendre langage HTML / créer soi-même ses éléments HTML (nom, vue, comportement)
Bilan : avantage (composants réutilisables dans une app – partageable entre projets / modèle arborescent des composants par imbrication des templates) / inconvénient (ajout de comportement)

PROGRAMMATION RÉACTIVE

+ Principe : approche visant à mieux gérer les flux (évènements discrets : frappe clavier / event continus ou comportement : position souris) / dépasser les callbacks ou le patron Observer
+ Pourquoi : gestion event et de l'async / faible latence / flux de donnes importants / tolérance aux fautes
+ Caractéristiques : responsive (reponse en temps voulu, temps de réponse rapides et fiables), resilient (resiste echec, echec n'impacte qu'un seul composant), elastique (système reste reactif en cas de variation de charge de travail), orienté message (passage de messages asynchrones, pas de blocages, composants consomment ressources quand ils peuvent) / immuabilité : objet dont etat ne peut pas être modifié après sa création (facilite prog fonctionnelle, une seule source de vérité, facilite le caching)
+ MVVM (Vue.js) / DOM Virtuel (model, DOM, virtual DOM, diff, patch)

PWA

+ Principes : fiable, rapide, engageant / Progressig : pour n'importe quel utilisateur indépendamment du navigateur (on améliore progresivement) / responsif : s'adapter au dispositif (indépendant de la connectivité, hors-ligne avec SW) / App-like : impression d'utiliser une application avec un modèle app shell / a jour : avec processus update SW / sûr : servie en HTTPS / découvrable : comme une app grâce au manifest et le registration scope des SW qui permettent aux moteur de recherche de les trouver, engageant avec notification push / installable / liable : facilement partageable avec url
+ Service Worker : proxy permettant de controller les requêtes http depuis vendant des pages web / pas acces au DOM, communication via passag de messages, mais accès à l'API IndexedDB
+ SW pour PWA : gestion notifications push (mécanisme de souscription au serveur pour recevoir des notifications, thread séparé) / mécanismes de caching variés

PROGRAMMATION MOBILE

+ Inconvénients (Taille de l'écran / Absence de clavier) / + Avantages (Tactile Nombreux capteurs / actionneurs, Matériel, Capacité de calcul) / Limiter la quantité de calculs côté client /Éviter les scripts trop gourmands /Privilégier les traitements côté serveur /Mémoire
+ Limiter (éviter l'inflation de) la taille des objets manipulés par les scripts
Ressources restreintes / Réseau : bande passante limitée (et payante)
Ne charger que les ressources nécessaires / Ne pas envoyer au client ce qui ne sera pas affiché

+ Réutiliser les ressources déjà présentes : /Bibliothèques de code (CDN)
+ Données statiques : cache du navigateur /Données utilisateur : Web Storage /Charger "intelligemment" les ressources
En asynchrone (pour ne pas bloquer l'interaction avec la page)
de manière "paresseuse" (lazy) : quand l'utilisateur en a besoin (attention au temps de chargement)
Rappel : vous pouvez utiliser les attributs HTML5 async et defer

EXPOSÉS	
Vue.js	**Angular**
+ Framework / JS / MVVM (Model View ViewModel) / gestion avancées états avec Vuex + Objectifs : coder API côté serveur et déporter logique applicative côté client / créer une app côté client from scratch / arborescence de composants + Spécificité : packager chaque composant dans un fichier .vue / + ecosystème plugins : plugins, store de gestion d'états, Vue CLI + Composants : définir échanges données à l'intérieur et entre les composants / mettre en place leurs comportement (en fonction cycle de vie) / lien modele – vue : one-way data binding et two-way data binding avec directives @click ou v-model) / structure app définie dans config du framework / propriétés : props, methods, computed / custom events + Templating : directives (v-), 2wbd / + Routing : vue-router / + State management pattern : Vuex	+ Framework / TypeScript / IoC (Inversion of Control) / Material design + Réécriture complète d'AngularJS + Utilisation TypeScript + Rétrocompatible avec JS + Objectifs : Scalabilité des applications web / Faciliter utilisation DOM / Implémenter tests complets /Routage, formulaires + Architecture MVC + DataBinding bidirectionnel + Injection de dépendances + La manipulation du DOM et moyen de directives **Svelte** Forces : Chargement rapide 2 Moins de code 3 Moins de débogage 4 Accessible 5 Réactive Faiblesses : Compatibilité 2 Moins conséquent 3 Petite communauté
Phaser	**Modernizr**
+ Game framework / JS ou TypeScript / WebGL, Canvas / moteur physique, sprites… **Ligthouse** + Browser plugin / NPM module / JS / analyse qualité – performance / outil reporting intégré **Node.js** + Plateforme d'exécution / JS / Event loop / Structure de modules arborescente (CommonJS)	+ Bibliothèque / JS / Détection des capacités d'un device – navigateur / Utilisation de classes de style CSS **PlayCanvas** + Game engine / JS / WebGL / Publication store – PlayCanvas – outil de collaboration **ESLint** + Linter – outil d'aide au développement / JS / Fonctionne à l'aide de règles (configurable) / Module NPM - CLI
React.js	**Babylon**
+ Bibliothèque / JS / Programmation réactive / Déclarative UIs + Bibliothèque JavaScript pour créer des interfaces utilisateurs / Programmation déclarative / À base de composants / Utilisable dans n'importe quel application et intégrable facilement + Principes : Composants / Fonctions ou classes ES6 / JSX De l'HTML dans le code JavaScript / State : stocker des données associées à un composant / Props : transmettre de l'information entre composants de manière unidirectionnelle / Hooks : fonctions permettant d'utiliser les fonctionnalités avancés d'un composant de type classe dans un composant de type fonction / DOM : arbre d'éléments HTML (texte), lent à parcourir / DOM virtuel : arbre de ReactElement (objets), rapide à parcourir / diff est réalisé entre les ReactComponents et les ReactElement du DOM virtuel afin de décider si une mise à jour est nécessaire	Moteur de rendu 3D basé sur WebGL / Écrit en Typescript / API Standard Javascript / Langage de shaders OpenGL / Shader : programme pour le rendu de la lumière sur des objets 3D / Interface bas niveau / Shaders auto-générés et adaptatifs / Comparaison avec Three.js / L'utilisation de couches de fallback. / Sur les mobiles, il utilise davantage le GPU que le CPU. / Seul Engine qui supporte WebGPU (expérimental dans Three.js). Avantages Babylon.js : - Editeur de text en ligne complet et intuitif - Support natif du TypeScript et de technologies récentes - Compatible nativement avec Unity, Maia, Blender, 3D Max, etc… Avantages Three.js : - Nettement moins lourd concernant la charge CPU et GPU - Nombreux exemples proposés par la communauté - Communauté plus nombreuse - Nombre de modules plus élevé
D3.JS	**Django**
Data-Driven Documents Bibliothèque graphique JavaScript ● Bibliothèque graphique JavaScript ● Construction de visualisations de données ○ Animations, Transitions ● Technologie SVG, CSS, HTML ● Compatible avec de nombreux formats de données Avantages ● Conforme au W3C ● Graphes dynamiques / Interactifs ● Types de fichiers pris en charges : JSON, CSV, TSV, XML, ... ● Bien documenté, grande communauté active avec énormément de tuto ● Compatibilité navigateurs modernes ● Open source Inconvénients ● Visualisation 2D uniquement ● Difficulté à maîtriser ● Performances avec de grandes données ⇒ Forte utilisation des ressources Importer la bibliothèque : <script src="https://d3js.org/d3.v6.min.js"></script> Ou ● Utiliser un gestionnaire de dépendance : npm install d3 **Firebase** • plate-forme de développement d'applications mobiles et Web • Beaucoup d'outils Alternatives open-sources ▪ Back4app ▪ Parse ▪ AWS Amplify ▪ Kuzzle ▪ Hoodies	Présentation générale : ■ Framework open-source en Python ■ Développé en 2003 pour le journal local de Lawrence(Kansas), par Adrian Holovaty et Simon Willison ■ Outil d'aide à la création d'application web, avec une base de données ainsi qu'une interface Avantages et utilisations : ■ Portabilité ■ Sécurité ■ Batteries-included ■ Scalabilité ■ Polyvalence Architecture MVT (Modèle, Vue, Template) Vue ■ Reçoit des requêtes HTTP et renvoie des réponses HTTP ■ Accèdent aux données requises pour satisfaire des requêtes via les modèles ■ Délèguent le formatage des réponses aux Template. Modèle ■ Objets pythons qui définissent la structure des données d'une application ■ Génère le SQL ainsi que le(s) ID nécessaire(s) ■ Permettent l'interaction avec une base de données en passant par un ORM ■ Mise à jour automatique de la base de données Template (gabarit) ■ Fait la mise en page de notre site ■ Les Balises de Template Django nous permettent de transférer le Python (provenant de la vue), en HTML ■ Notation dans les Template
CraftyJS	**Next.js**
PRÉSENTATION : GAME ENGINE • ENSEMBLE DE COMPOSANTS LOGICIELS POUR CONCEVOIR DES JEUX VIDÉOS. 3 • QUELQUES EXEMPLES DE JS GAME ENGINES : • CRÉÉ EN 2010 PAR LOUIS STOWASSER • DÉVELOPPÉ PAR TIM MARTIN, KEVIN SIMPER ET MUCAHO • FRAMEWORK POUR CRÉER DES JEUX VIDÉOS EN JAVASCRIPT • BASÉ SUR DES COMPOSANTS ET DES ÉVÈNEMENTS POUR LES JEUX EN JS • OPEN SOURCE • COMPATIBLE AVEC TOUS LES NAVIGATEURS • VERSION ACTUELLE : 0.9.0 INSTALLATION 5 npm install craftyjs bower install crafty • AVANTAGES : • LÉGER • OPEN SOURCE • GRANDE COMMUNAUTÉ • INCONVÉNIENTS : • TUTORIELS OBSOLÈTES • DOCUMENTATION PAS ASSEZ PRÉCISE ET PAS MISE À JOUR	Origines ➢ Un framework Web pour React ○ Front-end ○ open-source ○ JavaScript ➢ Développé par Vercel et la communauté Open-source ➢ Contribution de Google sur l'optimisation du framework en 2019 ➢ En 2020, NextJS a connu un grand succès ○ Utilisé par Netflix , Docker et GitHub etc.. Pourquoi NEXTJS ? ➢ Optimisation du rendu des pages ○ Diminue la charge de travail pour le navigateur ○ Génération des pages côté serveur ○ Optimisation SEO ○ Optimisation des images ➢ Outillages de développement pratiques : ○ Supporte TypeScript ○ Fast Refresh ○ Gestion des routes simplifiée
Phantom.JS	
Qu'est-ce que c'est ? • PhantomJS est un Headless browser → Navigateur sans GUI 1 • Basé sur Qt-WebKit • WebKit est un moteur de rendu développé par Apple • Qt-WebKit est un portage de ce moteur en utilisant Qt • Moteur de rendu → Chargé du parsing du DOM + CSSOM, JS • WebKit	

utilise: • WebCore pour le DOM / CSSOM • JavascriptCore pour le JS • WebKit est aujourd'hui utilisé essentiellement par Safari • PhantomJS propose de controler un navigateur sans tête a l'aide de JS → Le but va être de simuler un vrai utilisateur qui se promène sur l'internet mondial Avantages 3 • Rapide & léger • Parfait pour un environnement CI • Il existe des images Docker toute prêtes → utile pour les tests sur des runners • Embarque une API JS de génération d'évènements "user-like" • Quand PhantomJS simule un clic, c'est comme si un vrai utilisateur avait cliqué • Automatisable • On peut se déplacer dans l'historique des pages vues • Scraping web • Serveur NodeJS embarqué • Attention à la mauvaise gestion mémoire (cf. issue #14308) • OpenSource • Beaucoup d'utilisateurs... il y a quelques années ! Inconvénients 4 • Non maintenu ! • Problèmes de sécurité, de performance, etc. • Une très bonne alternative: pupeteer • De cet abandon résulte des problèmes de compatibilité avec différents formats • Cf #10839 • Absence de GUI → Fastidieux à configurer • Gestion de la mémoire • Il faut parfois rédémarrer complètement PhantomJS pour que la mémoire soit nettoyée, sous peine d'avoir une consommation de RAM exhorbitante et des crashes • N'échappe pas aux sécurités anti-bot • Captcha → nécessite de passer par un service tiers • Pas de support natif pour ES6 • Le support total d'ES2015 devait intervenir sur feu la version 2.5 (#14506)	**NuxtJS** Origines ➢ Un framework Web pour React ○ Front-end ○ open-source ○ JavaScript ➢ Développé par Vercel et la communauté Open-source ➢ Contribution de Google sur l'optimisation du framework en 2019 ➢ En 2020, NextJS a connu un grand succès ○ Utilisé par Netflix , Docker et GitHub etc.. Pourquoi NEXTJS ? ➢ Optimisation du rendu des pages ○ Diminue la charge de travail pour le navigateur ○ Génération des pages côté serveur ○ Optimisation SEO ○ Optimisation des images ➢ Outillages de développement pratiques : ○ Supporte TypeScript ○ Fast Refresh ○ Gestion des routes simplifiée
Framework de test	
Jest : - Open Source - Fonctionne notamment avec Babel, TypeScript, React, Angular.. - Utilisé par Facebook, Spotify, Instagram -Test runner -Tests exécutés en parallèles -API de base complète -Tests unitaires -back end et front end -JSDOM (Simule environnement DOM) Avantages - Rapidité (tests en parallèles) => plus rapide que Karma, Jasmine ou Mocha - Retour de tests lisibles - Test Coverage - Configuration simple (surtout avec React) - Syntaxe classique (compatible avec Jasmine) - Snapshots testing Inconvénients -Trés associé à React => manque de documentation pour l'utilisation avec d' autres frameworks sur certaines fonctionnalités. -Effets de bord possibles avec les tests effectués en parallèles => besoin d'avoir des tests strictement indépendant Jasmine : -créé en 2010 par Pivotallabs -open source -utilisable avec tous les frameworks -Tests unitaires Avantages Jasmine - Pas de dépendances - Pas de configuration - Grosse communauté => beaucoup de documentation - Peut être utilisé avec tout Framework classique - API compléte Inconvénients Jasmine -Pas le plus rapide -Pas de test snapshot (sans dépendance) -Se fait dépasser par Jest récemment Karma : Présentation Créer en 2012 Par Google - Lanceur de test automatique (a test runner) - Pour les test unitaire - Utilise des vrais navigateur - Open Source Avantages Créer en 2012 Par Google - Retour de tests lisibles - Rapide (autoWatch)	-Test sur des appareils réel -Se lance sur plusieurs navigateur simultanément - Framework de test agnostique - Configuration simple - Débogage facile - intégration continue Inconvénients - Limité aux test unitaires - Limité au Front-end Mocha : Présentation •Conçu pour tester à la fois du code synchrone et asynchrone •Soutenu par Clay global, Icons8, Sauce Labs,Localize… Créer en 2011 Open Source Avantages - Variété d'interface (BDD,TDD,Exports,QUnit,Require) - Fonctionne avec n'importe quelle bibliothèque d'assertion - Fonctionne pour le backend - Rapport de couverture de test. Inconvénients - Lent car les tests se font en mode série (les uns après les autres) Protractor : Présentation - Automatisation des tests - End to End - Angular - Test Integration - fonctionne avec d'autre framework jasmine,mocha, cucumber Avantages - Se lance sur plusieurs navigateur simultanément - Compatible avec plusieurs autre framework - Temps d'exécution des tests ? - Configuration (pour angular) Inconvénients - Débuggage - Documentation - Prise en main Comparaison Frameworks Mocha VS Jest VS Jasmine -Pour les tests unitaires -Mocha est modulaire => on choisit la/les librairies pour faire les assertions/mock/…. -Jest est le plus rapide des trois grâce à ses tests en parallèles

ANNALES	
Node.js	**Webpack**
+ Pas de Browser Object Model (BOM) car fourni par navigateur ; le code peut pas utiliser les API du navigateur tels que DOM, objet windows… + NPM rajouter version eslint dans package.json dans projet node : npm install eslint@^3.1.0 –save-dev + Node / Bower système de résolution de dépendances : Node (côté serveur -> résolution des dépendances, pas de problème de place -> sous-dépendances isolées -> dépendances arborescentes) / Bower (côté client -> résolution ded dépendances au chargement, limitation de la bande passante et du stockage -> factorisation des sous-dépendances -> « à plat »	+ Types de fichiers packagés dans bundle : js, json, css, image, police… + Raisons performance packaging : fichiers minifiés / une seule transaction http pour les obtenir + Problème débugage app générée : code minifié donc difficile de trouver fichiers, lignes sources dans les devs tools à moins de mapper code source sur code minifié **APP** + Service workers : permet de contrôler le câche de l'application (app shell et données), ce qui permet de redémarrer application sans la recharger ou en offline, comme installée en local (PWA)
Vue.js	**Navigateur / Mobile**
+ Fichier .vue : template (HTML) /script (JS) / style (CSS) du composant + Router : associe route côté client (hash URL) à composant / se configure avec objet JSON spécifiant ces associations et d'éléments HTML « router-link » indiquant où placer les composants templatés + Custom Events : utilisé pour échanger messages entres composants éloignés dans arborescence + Vuex : mutations synchrones et actions asynchrones + Watcher : écouter notification changement du modèle ; pousser changements aux directives concernées ; gérer dépendances entres éléments du modèle (changement variable peut déclencher celui d'une autre) + Vuex / LocalStorage : LocalStorage persistant et pas Vuex	+ Persister information côté client : Cookie (petites quantités de données, liées à la session) / SessionStorage (grandes quantités de données, liées à la session) / Service worker (cache de données et de code applicatif) + Device APIs HTML5 : demande à l'utilisateur autorisation pour Battery, Media capture, Geolocation, Notification + Multitouch : détecter en s'abonnant à évènement touchstart et utiliser attribut length de la TouchList touches + Mobile First : utilisation media queries de manière « ascendante » (ex : p.details {visibility: hidden;} @media screen and (min-width:500px) {p.details {visibility: visible;}}) - Mobile First : réduit temps de dev app web car limitation des fonctionnalités développées : pas de re-développment du site mobile

Web	Mécanismes
+ WebSocket : préférable à requêtes-réponses http car envoie de données « streamées » depuis capteur de position, push de nouveau messages d'un serveur de chat à ses clients + LocalStorage / AppCache : permettent de réaliser des applications web offline	+ One-way data-binding : sens données vers interface ; permet de garder trace des changements appliqués sur les données, meilleure gestion asynchrone et concurrence, pas synchronisation « magique » + data-flow / templating « classique » : avantage data-flow évite les problèmes de boucle en two-way-binding ; inconvénient : nécessite de définir des états pour le contrôle de l'interface
JS	
+ Event listeners et callbacks : par réellement faire du multitâche car event listener rajout un message à la file demandant l'exécution du callback / son exécution commencera quand event loop aura exécuté toutes tâches insérées dans la file précédemment et se fera avant toute tâche insérée ultérieurement	+ Pattern promesse et notion de fonction d'ordre supérieur : une fonction d'OS peut recevoir en paramètre une autre fonction -> callbakces resolve & reject passés à la construction d'une promesse. + Inférence de type : intérêt dynamicité : inconvénient performance

1. Quel est le principal problème que l'on peut avoir pour débugger une application Web générée avec WebPack et comment s'en prémunir ? Pourquoi peut-on avoir le même problème avec Web Assembly ? (2 pts.)
   Le code est minifié et il est donc impossible de trouver, en regardant les dev tools, quels sont les fichiers / lignes des sources où se trouvent les problèmes… À moins de mapper le code source sur le code minifié. De la même façon, les modules Web Assembly sont compilés et chargés côté client en bytecode.
2. Listez et expliquez à quoi servent les 3 grandes fonctions du Watcher de VueJS. (2 pts.)
   Écouter les notifications de changements du modèle ; pousser ces changements aux directives concernées ; gérer les dépendances entre éléments du modèle (le changement d'une variable peut déclencher celui d'une autre).
3. Expliquer dans quel sens se fait le one-way data binding, et pourquoi il peut être intéressant de faire du one-way data-binding plutôt que du two-way data-binding ? (2 pts.)
   Sens : données vers interface ; permet de garder la trace des changements appliqués sur les données, meilleure gestion de l'asynchrone et de la concurrence. Pas de synchronisation « magique ».
4. Donnez un exemple de code malicieux que pourrait renvoyer le serveur suite à une requête cross-domaine en JSONP et expliquez le processus qui fait que ce code serait forcément exécuté côté client. (2 pt.)
   N'importe quel code JS (par exemple window.close();) peut être exécuté, puisque ce code se retrouverait dans un élément script de la page, et donc interprété par le navigateur.
5. Dans l'approche « Mobile First », on recommande l'utilisation des media queries de manière « ascendante ». Donnez un exemple pour illustrer ce point. (2 pts.)
   ```
   p.details {visibility : hidden;}
   @media screen and (min-width: 500px) {
   p.details {visibility : visible;} }
   ```
6. Expliquer techniquement pourquoi l'utilisation d'event listeners et de callbacks ne permet pas réellement de faire du multitâche. (2 pts.)
   L'event listener rajoute un message à la file demandant l'exécution du callback. Son exécution commencera lorsque l'event loop aura exécuté toutes les tâches insérées dans la file précédemment et se fera avant toute tâche insérée ultérieurement.
7. Expliquez ce qu'apportent les service workers aux Progressive Web Apps et à quel besoin ils répondent en comparaison aux applications natives. (2 pt.)
   Il permet de contrôler le cache de l'application (app shell et données), ce qui permet de redémarrer l'application sans la recharger ou en offline, comme lorsqu'elle est installée en local.
8. Écrivez la commande NPM qui permet de rajouter la version 3.1.0 (ou une version mineure supérieure) d'eslint au bon endroit dans le fichier ./package.json de votre projet Node. (2 pt.)
   ```
   npm install eslint@^3.1.0 --save-dev
   ```
9. Quelle est la différence entre le store Vuex et LocalStorage (1 pt.)
   L'un (LocalStorage) est persistant et pas l'autre.

Outils (4 points)

10. Durant les cours et les exposés, un assez grand nombre d'outils ont été abordés. Remplissez le tableau ci-dessous (la première ligne est un exemple) :

Nom	Type d'outil	Langage principal	Spécificité technique 1	Spécificité technique 2
Vue	Framework	JS	MVVM	Gestion avancée des états avec Vuex
Angular	Framework	TypeScript	IoC	Material design
Phaser	Game framework	JS ou TypeScript	WebGL, Canvas	Moteur physique, sprites…
React	Bibliothèque	JS	Programmation réactive	Declarative UIs
Lighthouse	Browser plugin / NPM module	JS	Analyse de qualité / performance	Outil de reporting intégré
PlayCanvas	Game engine	JS	WebGL	Publication sur le store PlayCanvas / outils de collaboration (payants)
Node	Plateforme d'exécution	JS	Event loop	Stucture de modules arborescente (CommonJS)
Modernizr	Bibliothèque	JS	Détection des capacités d'un device / navigateur	Utilisation de classes de style CSS
ESLint	Linter / outil d'aide au développement	JS	Fonctionne à l'aide de règles (configurable)	Module NPM / CLI

1. Pourquoi `Node.js` ne comporte-t-il pas de `Browser Object Model` (BOM), et qu'est-ce que cela implique pour le code tournant dans cette plateforme ? (1 pt.)
   Le BOM est fourni par le navigateur ; le code ne peut donc pas utiliser les API du browser tels que le DOM, l'objet window, etc.
2. Citez 4 types (génériques) de fichiers qui peuvent être packagés dans un `bundle webpack`. (2 pts.)
   js, json, css, image, police…
3. Indiquez 2 raisons pour lesquelles le fait de packager des fichiers dans un `bundle webpack` permet de gagner en performance au moment du chargement de l'application. (1 pt.)
   Les fichiers sont minifiés et il y a une seule transaction HTTP pour les obtenir.
4. Que trouve-t-on dans un fichier `.vue` représentant un composant `VueJS` ? (1 pt.)
   le template, le code JS et le CSS de ce composant.
5. Indiquez brièvement comment s'utilise le `Router` de `VueJS`. (2 pts.)
   Il associe une route côté client (hash de l'URL) à un composant. Il se configure à l'aide d'un objet JSON spécifiant ces associations et d'éléments HTML router-link indiquant où placer les composants templatés.
6. Donnez un exemple d'utilisation des `Custom Events` dans `VueJS` dans une application. (1 pt.)
   Échanger des messages entre composants éloignés dans l'arborescence.
7. Quelle est la différence principale entre `Actions` et `Mutations` dans un `store Vuex` ? (1 pt.)
   Les mutations sont synchrones et les actions peuvent être asynchrones.
8. Citez 4 moyens de persister de l'information côté client et indiquez une caractéristique de chacun d'eux (2 pts.)
   Cookie : petites quantités de données, lié à la session
   SessionStorage : grandes quantités de données, lié à la session
   LocalStorage : grandes quantités de données, indépendant de la session
   Service worker : cache de données et de code applicatif
9. Citez 2 noms de `Device APIs HTML5` pour lesquelles le navigateur demande à l'utilisateur l'autorisation de les exécuter. (1 pt.)
   Battery, Media capture, Geolocation.
10. Indiquez (par du texte ou du code) comment détecter le `multitouch` dans un navigateur (1 pt.)
    S'abonner à l'événement touchstart et utiliser l'attribut length de la TouchList touches.
11. Citez un cas d'utilisation précis pour lequel il est préférable d'utiliser des `WebSocket` plutôt que des requêtes-réponses `HTTP`. (1 pt.)
    Envoi de données « streamées » depuis un capteur de position, push de nouveaux messages d'un serveur de chat à ses clients.
12. Expliquez, en `WebAssembly`, l'enchaînement des 2 promesses à utiliser suite à la commande `fetch`. (2 pts.)
    ```
    let wasmProm = fetch (ressource serveur) ;
    instantiateStreaming (wasmProm) // Crée l'instance à partir des résultats de la promesse précédente
    .then (results) => { traitements }
    ```