WealCome
Expertise en conception logicielle
BDD / TDD / DDD / Clean Architecture

PROGRAMME DE FORMATION

Document généré le 23/11/2026

Agents IA et Software Craft : la vitesse sans la dette — Monde Angular / TypeScript

Type d'action : **Action de formation** Durée totale : **14h**

Informations de session

Lieu : Visio par Zoom

Formateur : Michaël AZERHAD - Président de S.A.S.U WealCome et architecte / expert technique depuis plus de 20 ans

Dates et horaires

Date	Début	Fin	Durée
09/12/2026	09:15	12:15	3h
09/12/2026	13:00	17:00	4h
10/12/2026	09:15	12:15	3h
10/12/2026	13:00	17:00	4h

Description

Tout le monde produit vite avec l'IA. Peu de gens produisent vite, et longtemps.

Sur un frontend Angular, l'IA livre en quelques minutes des écrans entiers, des services et des stores. Sans Software Craft, ce qu'elle produit ne tient pas la distance :

- une logique métier noyée dans les composants, les templates et des services fourre-tout ;
- un état éparpillé entre services, signaux et appels HTTP, que plus personne ne maîtrise ;
- des tests écrits après coup, qui rassurent sans réellement protéger ;
- du code en trop, des hallucinations, des fonctionnalités que personne n'a demandées ;
- un contexte qui sature et des tokens qui explosent en allers-retours ;
- du code relu ligne à ligne, sans savoir vraiment le juger ;

un projet qui bascule en legacy en quelques mois au lieu de quelques années.

Cette formation prend le problème à la racine.

En deux jours, vous menez un vrai projet de bout en bout : une spécification brute, mal ficelée, telle qu'un fondateur vous la donnerait, jusqu'à un front Angular modulaire où plusieurs features tournent ensemble. Ce sont les agents IA qui produisent ; c'est vous qui décidez.

Technologies :

- Angular 22+
- TypeScript, version courante
- Signaux Angular (branche signaux seuls)
- NgRx Signal Store et son plugin Events
- Vitest / Angular Testing Library
- Stryker (mutation testing)
- Purify-ts, Effect-TS (gestion d'erreurs typée)
- Claude Code : agents et skills conçus par Michaël AZERHAD

Objectifs de la formation

À l'issue de la formation, les participants auront acquis les connaissances et méthodes nécessaires pour :

- Comprendre les principes du Software Craft : agilité, NoEstimates, DDD, BDD, TDD, refactoring et Clean / Hexagonal Architecture.
- Maîtriser les stratégies de tests automatisés (unitaires, intégration, end-to-end, mutation testing avec Stryker) et les principes de qualité logicielle.
- Comprendre les spécificités du DDD appliqué au frontend Angular : langage ubiquitaire, Bounded Contexts et responsabilités front / back.
- Maîtriser les principes d'une architecture Angular modulaire et testable : use cases, Domain Model, View Models, adaptateurs et composants « humble objects ».
- Comprendre les approches de gestion d'état avec les signaux Angular, NgRx Signal Store et le plugin Events, ainsi que leurs compromis.
- Appréhender la programmation événementielle, la distinction état serveur / état client, les optimistic updates, les flux asynchrones et l'optimisation de la détection des changements.
- Comprendre les stratégies de gestion des erreurs, de navigation et de modularisation du frontend.
- Maîtriser les principes d'orchestration des agents IA, de la spécification BDD à la livraison d'une application Angular exécutable.
- Comprendre les méthodes de supervision, de configuration et de contrôle qualité des agents IA pour limiter hallucinations, sur-implémentation et dette technique.
- Développer une approche critique des choix architecturaux et savoir promouvoir les pratiques Software Craft au sein d'une équipe.

Prérequis

- Solide expérience du développement (au moins 5 ans recommandés)
- Bonne maîtrise de TypeScript et d'Angular
- Notions de state management appréciées (NgRx ou autre)
- Bonnes connaissances en programmation orientée objet et/ou fonctionnelle
- Capacité à écrire un simple test unitaire avec Jest ou Vitest
- Aucune expérience préalable des agents IA requise. « Niveau expert » désigne le niveau de pratique visé, pas un prérequis.

Public visé

Particuliers et professionnels :

- CTO / Technical Leader
- Développeur FrontEnd
- Développeur Full Stack
- Architecte technique

Méthodes pédagogiques

- Live coding et orchestration d'agents en direct par le formateur
- Mob programming : les participants codent et orchestrent ensemble
- Apports théoriques émergeant de la pratique, sur un cas métier réaliste
- Comparaisons entre code écrit à la main et code produit par les agents, et entre signaux seuls et NgRx Signal Store
- Challenges proposés sur le TDD, la Clean Architecture, le state management et le refactoring

Les agents Claude Code du formateur sont utilisés tout au long des deux jours, et non cantonnés à une démonstration.

Selon le curseur IA choisi par le groupe, les participants observent, puis orchestrent eux-mêmes les agents : spécification, tests, code, validation par les tests et refactoring.

Capacités développées

À l'issue de la formation, les participants seront capables de :

Axe 1 — Software Craft

- Identifier le « produit dans le produit » et découper les besoins en itérations courtes selon NoEstimates.
- Appliquer les principes du DDD au frontend Angular : langage ubiquitaire, parcours entre Bounded Contexts et délimitation des responsabilités front / back.
- Rédiger des spécifications BDD non ambiguës, centrées sur les comportements plutôt que sur l'interface et exploitables comme tests d'acceptation.
- Pratiquer le TDD Outside-In sur Angular, en partant des composants ou des use cases.
- Concevoir des tests unitaires, d'intégration des adaptateurs et end-to-end in-memory avec Vitest et Angular Testing Library ; distinguer tests sociables / solitaires et utiliser les doublures adaptées.
- Appliquer le mutation testing avec Stryker, comprendre son lien avec le TDD et viser un score de 100 % pour limiter le code superflu.
- Refactorer en continu le code de production et les tests sans régression.
- Concevoir un Domain Model frontend, identifier ses défauts et choisir entre un modèle global ou par use case.
- Structurer une application Angular en Clean / Hexagonal Architecture : use cases, Domain Model, View Models pilotés par les tests (withComputed avec NgRx Signal Store, presenters avec signaux seuls), états d'écran et composants « humble objects ».
- Concevoir et interchanger les adaptateurs secondaires : backend en mémoire, stockage navigateur et API tierces.
- Comparer les stratégies de gestion d'état avec les signaux Angular seuls ou NgRx Signal Store, choisir la plus adaptée et positionner les use cases.
- Mettre en œuvre une architecture événementielle avec le plugin Events de NgRx et la comparer aux appels directs entre services utilisant les signaux.
- Distinguer état serveur et état client, et identifier les responsabilités de httpResource() et de NgRx Signal Store.
- Exploiter la granularité des signaux pour limiter les détections de changements inutiles.
- Implémenter en TDD des optimistic updates et des flux asynchrones.
- Organiser un frontend modulaire avec un routing piloté par les use cases, indépendamment du découpage métier du backend.
- Gérer les erreurs avec des types Either ou des exceptions, en évaluant leurs compromis jusqu'à l'affichage des états d'erreur.

Axe 2 — Développement agentique

- Orchestrer des agents IA de la spécification à l'application exécutable, avec supervision des décisions ou délégation en mode « trust ».
- Produire des spécifications BDD structurées, catégorisées, séquencées et suivies par statut de complétion.
- Superviser le TDD réalisé par les agents et prévenir hallucinations, code superflu et sur-implémentation.
- Contrôler la qualité du code généré sans relecture systématique ligne à ligne, grâce aux spécifications, tests, mutation testing et règles architecturales.
- Évaluer, corriger et justifier les choix de conception proposés par l'IA selon les principes du Software Craft.
- Configurer les agents, commandes « décorateurs » et hooks pour fiabiliser les workflows.

- Faire générer les environnements de développement, configurations et jeux de données par story.
- Selon les options retenues : refactorer avec les agents un frontend legacy généré par IA vers une architecture testable et maintenable.

Capacités transversales

- Argumenter et promouvoir les pratiques Software Craft auprès d'une équipe, y compris auprès de profils sceptiques.
- Concilier productivité apportée par l'IA, qualité logicielle, maintenabilité et maîtrise des décisions techniques.

Modalités d'évaluation

- QCM en fin de formation
- Mise en pratique : les participants pilotent eux-mêmes les agents sur le projet fil rouge
- Échanges et discussions

Documents et supports pour les apprenants

- Supports de cours en markdown très complets
- Les agents et skills Claude Code du formateur (orchestrateur, TDD, spécification BDD et bien d'autres), rodés sur plus de 200 000 lignes de code en production
- Projet Git du live coding avec ses deux branches : signaux seuls et NgRx Signal Store
- Partage de ressources (articles, blogs, discussions de forums tech)
- Discussions privées sur le canal Slack dédié au mini-groupe post-formation

Déroulé de la formation

Jour 1 — Spécifier et livrer des features Angular multi-scénarios

Matin — Du besoin brut à la première feature en TDD

- Tour de table : parcours de chacun, attentes précises et objectifs de formation.
- Présentation du projet fil rouge : le frontend Angular d'un projet complexe, digne des grands produits tech, qui sert de support à toute la formation. Il concentre à la fois les défis du Software Craft et ceux du développement agentique, avec tous leurs pièges et leurs astuces. Le backend est simulé en mémoire, pour se concentrer pleinement sur le front. L'interface visuelle est fournie : la formation porte sur le comportement, pas sur le style.
- Quiz oral express pour jauger le niveau de chacun sur les fondamentaux du craft, avant même d'attaquer le projet. Des questions choisies pour provoquer des déclics immédiats et éveiller la curiosité.
- L'agilité, clarifiée : une mise au point rapide sur ce qu'est vraiment l'agilité. L'ambition de la formation est de la pousser à l'extrême grâce au développement agentique, avec des itérations très courtes et des spécifications intelligentes. L'objectif est de passer d'une logique projet à une logique produit.
- NoEstimates, notre fer de lance pendant tout le projet : ce n'est pas ce que l'on croit, mais une pratique profondément agile, qui existait bien avant l'IA.
- Une spécification brute, telle qu'un fondateur la donnerait : un projet d'envergure aux specs mal travaillées, comme on en reçoit au quotidien. On discute avec les participants de la façon de l'aborder. On déconstruit les « bonnes pratiques » courantes qui sont en réalité de mauvaises pratiques, puis on bascule vers les pratiques enseignées.
- Trouver le produit dans le produit avec des agents agilistes spécialisés : réduire le périmètre pour livrer d'abord la valeur la plus petite mais la plus impactante. On mène une discussion avec le métier, et l'IA projette au moins 5 orientations business, chacune avec sa propre plus-value. L'humain choisit celle qui définit le premier produit.
- Le DDD vu depuis le front : pourquoi la modélisation stratégique et tactique du DDD vit surtout côté back (agrégats, invariants transactionnels, cohérence entre contextes), et ce que le front en garde vraiment : le langage ubiquitaire, du BDD jusqu'au code ; la lecture des Bounded Contexts qu'un même parcours utilisateur traverse, et la façon dont le front les fait communiquer ; une modularisation du front qui ne se calque pas forcément sur les Bounded Contexts.
- Atelier BDD express sur un premier parcours utilisateur, animé par un agent : il pose des questions pointues sous forme de QCM, plus rapides qu'un atelier classique entre humains, et fait émerger les comportements attendus du front. On évite le piège des scénarios qui décrivent l'interface (« je clique sur le bouton… ») plutôt que le comportement. Les spécifications sont produites dans un format optimisé pour l'IA : catégorisées, séquencées, numérotées, avec des flags de statut de complétion.
- Le TDD ne fait pas la macro-architecture : son rôle, ce sont les détails d'implémentation. Les grandes décisions se prennent avant lui. Le TDD s'inscrit ensuite dans le découpage en features et en parcours issu de la spécification.
- Introduction avancée à la Clean Architecture côté front : de quoi apprendre de nouvelles choses, même aux plus seniors. Use cases, Domain Model, View Models et adaptateurs : le composant Angular devient un simple « humble object ». On voit comment elle resserre la sphère d'action de l'IA, et on casse une fausse croyance : l'hexagonal n'impose pas de tout repousser en périphérie. Certaines décisions, comme le choix d'un state manager tel que NgRx Signal Store, ont toute leur place au cœur de l'architecture.

- Test unitaire, test d'intégration, test end-to-end : ce n'est pas ce que l'on croit. On clarifie ce que recouvre vraiment chaque type de test dans le monde du front, et sa place dans la démarche ATDD / TDD : tests unitaires : Vitest ; tests d'intégration : les adaptateurs secondaires, avec Vitest ; tests end-to-end in-memory : depuis les composants Angular, avec Angular Testing Library.
- Par où commencer la technique ? Composants d'abord ou comportements d'abord : on en débat, et on casse les idées reçues. On attaque la première feature en live avec une démarche ATDD / TDD revisitée pour l'IA : l'intérêt du TDD, et ce qu'il apporte avec l'IA ; les déboires sans vrai TDD : hallucinations, code en trop, code qui va au-delà de l'attendu ; le cycle red – clean green but minimal – maybe refactor further ; l'Outside-In et ses deux variantes côté front : en partant des composants Angular, ou en partant de la logique métier (use cases) ; les tests sociables, le programming by wishful thinking ; les agents TDD de Michaël : ce qui change, et pourquoi c'est si efficace.
- TDD et mutation testing (Stryker) : le lien fort entre les deux. On voit pourquoi un vrai TDD rend le score de mutation testing à 100 % presque naturel, et pourquoi c'est le meilleur garde-fou contre le code superflu produit par l'IA.

Après-midi — De la première feature opérationnelle à plusieurs features

- Règles de gestion, de la plus simple à la plus complexe, pilotées par les agents, use case par use case. On voit tous les rouages du refactoring et les astuces pour travailler avec l'agent. On voit aussi comment l'IA garde le cap à mesure que le logiciel évolue, en maintenant un mutation testing à 100 %.
- Le Domain Model du front émerge des tests : par use case ou global, ce que chaque option implique, la faille que tout le monde commet, et comment les agents orientent vers la bonne conception au fil du refactoring, sous validation humaine.
- Avec ou sans state manager : le groupe choisit à la majorité la branche construite en live, puis on présente l'autre approche : une branche signaux seuls : des services et des signaux Angular, sans state manager. Les signaux sont une primitive de réactivité, pas un state manager ; une branche NgRx Signal Store avec le plugin Events. Le formateur termine les implémentations des deux branches pour le lendemain, afin de les comparer en détail.
- La génération de View Models, dirigée par les tests : un point clé de la Clean Architecture côté front. Avec NgRx Signal Store, elle passe par withComputed. Avec les signaux seuls, par des presenters classiques. Les View Models portent aussi les états d'écran (chargement, erreur, vide), là où les fronts générés par l'IA sont les plus fragiles.
- Premiers adaptateurs secondaires au sens Clean Architecture : le backend simulé en mémoire, qui reproduit latence et échecs pour rendre testables les cas d'erreur, le stockage navigateur (localStorage, IndexedDB) et une API tierce, le tout verrouillé par des tests d'intégration.
- Adaptateur primaire (les composants Angular) et super-orchestrateur : un niveau au-dessus des agents TDD, il décompose les problèmes et enchaîne plusieurs tâches macro (adaptateurs, logique métier, branchement des composants…) en déléguant l'itération aux agents. Il propose aussi un mode « trust », qui laisse les agents enchaîner sans validation intermédiaire. Le prompt manuel devient marginal, et le développeur se concentre sur les décisions qui comptent. Démonstration du remote-control de Claude Code : on pilote les agents et on fait avancer tout le développement depuis un simple téléphone.
- Une feature entièrement fonctionnelle : les agents lancent l'application Angular en mode dev, branchée sur le backend en mémoire. Ils génèrent des jeux de données par story pour parcourir chaque scénario directement dans le navigateur. On bascule entre implémentations en mémoire et réelles (stockage navigateur, API tierce) grâce aux configurations d'environnement d'Angular. Des agents de configuration génèrent l'environnement et des scripts de configuration simples.

- Plusieurs nouvelles features, proposées par les participants : une fois la première feature livrée, les participants soumettent de nouvelles features du produit. Chacune passe par un atelier BDD express avec l'agent, puis le super-orchestrateur les livre l'une après l'autre, dans la branche choisie par le groupe. On voit l'architecture absorber chaque nouvelle feature sans se dégrader, avec un mutation testing maintenu à 100 %. Les participants qui le souhaitent prennent la main, en partage d'écran.
- Clôture du jour 1 : questions-réponses, puis partage des supports de cours qui reprennent par écrit tout ce qui a été vu, enrichis d'illustrations générées par l'IA à partir du code réellement produit.

Jour 2 — Passer à l'échelle avec un front modulaire

Matin — Conception avancée

- Tour de table : ce qui a été compris, ce qui l'est moins.
- Comparaison des branches, signaux seuls ou NgRx Signal Store : les mêmes features, terminées par le formateur dans chacune des branches. On voit où vivent les use cases selon la branche, et pourquoi : avec NgRx, le store est au cœur de l'application (withReducer, withEventHandlers) ; avec les signaux seuls, les use cases vivent dans un noyau TypeScript pur. On compare les View Models produits par withComputed et par des presenters, et ce que NgRx apporte réellement par rapport aux signaux seuls. On montre l'impact sur la détection de changements et les performances (granularité des signaux, computed, mode zoneless).
- Avec le plugin Events de NgRx, on comprend aussi la programmation événementielle, inspirée de Flux : un événement décrit ce qui s'est passé, et plusieurs stores peuvent y réagir sans se connaître. Le quoi est découplé du comment, bien au-delà de ce qu'offrent les signaux seuls, où les services s'appellent directement.
- On voit aussi les intérêts subtils, côté testabilité, d'un state manager placé au cœur : un use case se teste sur un vrai store, sans mock ni wrapper : le test reste sociable et résiste aux refactorings ; le Given devient un simple état initial et le Then une assertion sur l'état du store, au plus près des scénarios BDD ; les dépendances (gateways) sont injectées dans le store, par l'injection de dépendances d'Angular : chaque cas d'erreur ou de latence devient testable avec de simples fakes ; aucun port artificiel n'est nécessaire pour abstraire le store : moins d'indirection, et moins de code sur lequel l'IA peut dériver.
- Routing et navigation : une navigation pilotée par les use cases, testable hors des composants.
- Le Domain Model, en profondeur : des micro-exercices de modélisation, avec et sans IA. On apprend à juger les choix de l'IA grâce au craft, à savoir quand la contredire et la recadrer, et à la faire progresser par ses propres connaissances. Le but est de ne plus subir les propositions de l'IA, mais de dialoguer d'égal à égal avec elle.
- État serveur, état client et optimistic updates : ce qui relève de l'état serveur et ce qui relève de l'état client, et pourquoi les confondre crée de la complexité accidentelle. Avec les signaux seuls, resource() et httpResource() d'Angular couvrent la lecture de l'état serveur ; NgRx Signal Store, lui, réunit état serveur et état client dans un même store. Ni l'un ni l'autre n'offre de cache serveur à la manière de TanStack Query : on voit ce que cela implique pour l'invalidation et la cohérence des données. On met ensuite à jour l'interface avant la réponse du serveur, et on revient en arrière proprement en cas d'échec, en TDD avec les agents. Le backend en mémoire simule latence et échecs, ce qui rend chaque cas testable.
- Orchestration asynchrone : enchaîner plusieurs use cases et appels selon la branche choisie. Avec NgRx, withEventHandlers fait de cet enchaînement une réaction à des événements, et non une suite d'appels : c'est l'équivalent front des Process Managers du back.
- Modularisation du front : pourquoi un deuxième module émerge, comment le front fait communiquer les Bounded Contexts qu'un parcours traverse, et pourquoi calquer ses modules sur les Bounded Contexts du back est un piège. Avec NgRx, deux modules communiquent par événements sans dépendre l'un de l'autre, un écho direct au messaging entre Bounded Contexts côté back. Les événements à portée limitée (scoped events) isolent ce qui doit rester propre à une feature.

Après-midi — Maîtrise des agents : les participants aux commandes

- Commandes « décorateurs » et hooks : paramétrer un agent sans toucher à sa description, et renforcer le cadrage avec les hooks.
- Réalisation d'une feature en autonomie par les participants, qui sert d'évaluation des acquis. Ils implémentent les règles de gestion, la navigation et les View Models propres à la feature en pilotant eux-mêmes les agents de Michaël depuis leur Claude Code (ou équivalent), dans la branche de leur choix, signaux seuls ou NgRx Signal Store, selon ce qui les concerne au quotidien. L'objectif est une production très rapide, dans un cadre normé.
- Gestion d'erreurs : types Either ou exceptions classiques : on compare les deux styles dans le cœur de l'application. D'un côté, des erreurs explicites dans les signatures (types Either, avec des librairies comme Purify-ts ou Effect-TS), que le compilateur oblige à traiter ; de l'autre, les throw / try…catch classiques de JavaScript, plus familiers mais invisibles dans les types. On suit l'impact jusqu'aux View Models qui affichent les états d'erreur. Ce que chaque approche apporte, ses pièges, et la façon dont les agents s'adaptent au style retenu.
- Discussions approfondies sur des thèmes choisis par les participants : micro-frontends face au front modulaire, les mauvaises pratiques de l'agilité en entreprise… pour ancrer la formation dans la réalité de l'entreprise.
- Convaincre ses collègues : arguments et conseils pour introduire ces pratiques dans son équipe, y compris auprès des adeptes du « vibe coding » qui n'en voient pas encore l'intérêt.
- Clôture : QCM final, auto-évaluation des capacités développées, installation des agents dans l'environnement de chacun et mise en place du canal Slack privé de suivi post-formation. Remise du projet Git avec ses deux branches : signaux seuls et NgRx Signal Store.
- En option : à un moment donné de la formation, intégrer une large portion de code legacy front, générée par l'IA sans aucune fondation craft (logique dans les composants, services fourre-tout…). On voit alors comment les agents permettent de traiter ce legacy très facilement, très rapidement et de manière très cadrée.

Modalité d'accès

Contactez-nous par mail : contact@wealcomecompany.com

Délais d'accès

Nous pouvons programmer les formations en fonction de vos contraintes et de nos disponibilités.

Moyens d'accompagnement et d'assistance (Formation à distance)

En cas de difficulté d'accès à la plateforme de visioconférence ou pour toute question technique ou pédagogique pendant la session, le stagiaire dispose d'un point de contact unique pour obtenir une assistance immédiate :

Contact d'assistance technique et pédagogique : WealCome

E-mail : contact@wealcomecompany.com

Accessibilité handicap

Nos formations sont accessibles aux personnes en situation de handicap.

Les aspects, l'accessibilité et le type de handicap au regard des modalités pédagogiques sont à évoquer impérativement au cours de l'entretien préalable à toute contractualisation afin de pouvoir orienter ou accompagner au mieux les personnes en situation de handicap.