Jhipster: JHipster offre agli sviluppatori una piattaforma per creare facilmente applicazioni web e microservizi da zero, senza dover passare molto tempo a collegare insieme tutto e integrare le tecnologie.

**Introduzione allo sviluppo di applicazioni Web moderne**

L'industria del software definisce uno sviluppatore full-stack come qualcuno che può lavorare su diverse aree di uno stack di applicazioni. Il termine pila si riferisce a diversi componenti e strumenti che costituiscono un'applicazione.  
In termini di sviluppo di applicazioni Web, lo stack può essere ampiamente classificato in due aree: stack frontend e backend o stack lato client e server side. Il frontend si riferisce generalmente alla parte responsabile del rendering dell'interfaccia utente, mentre il backend si riferisce alla parte responsabile della business logic, delle interazioni del database, dell'autenticazione dell'utente, della configurazione del server e così via. Uno sviluppatore di applicazioni Web Java full stack dovrebbe lavorare su entrambe le tecnologie di frontend e backend, che vanno dalla scrittura di HTML / JavaScript per l'interfaccia utente alla scrittura di file di classe Java per la logica aziendale e query SQL per le operazioni di database come richiesto.

Con un panorama di architettura software in continua evoluzione, l'ambito delle tecnologie che uno sviluppatore web full-stack dovrebbe funzionare è aumentato enormemente. Non è più sufficiente scrivere HTML e JavaScript per creare un'interfaccia utente, ci si aspetta che conosca framework lato client come Angular, React, VueJS e così via. Inoltre, non siamo abbastanza esperti in Java e SQL, ci aspettiamo di conoscere framework lato server come Spring, Hibernate, Play e così via.

***Transpillers:*** I transpillers sono compilatori sorgente-sorgente. Mentre un compilatore tradizionale viene compilato da sorgente a binario, un traspiler compila da un tipo di codice sorgente a un altro tipo di codice sorgente. TypeScript e CoffeeScript sono eccellenti esempi di questo, entrambi compilano fino a JavaScript.  
È molto facile perdersi qui e qui è dove tecnologie come JHipster e Spring Boot intervengono per aiutare. Vedremo i dettagli nei capitoli successivi, ma in breve, aiutano a fornire il cablaggio tra le parti in movimento in modo che sia sufficiente concentrarsi sulla scrittura del codice commerciale. JHipster aiuta anche fornendo le astrazioni per distribuire e gestire l'applicazione a vari fornitori di cloud.

***Modelli di architettura Web***

I modelli di architettura delle applicazioni web ampiamente utilizzati oggi possono essere classificati in architettura monolitica e architettura a microservizi.

***Architettura web monolitica***

un'applicazione web monolitica può fare quanto segue:  
• Può supportare diversi client come browser desktop / mobile e applicazioni native desktop / mobile  
• Può esporre le API per il consumo di terze parti  
• Può essere integrato con altre applicazioni su servizi Web REST / SOAP o code di messaggi  
• Può gestire richieste HTTP, eseguire la business logic, accedere a un database e può scambiare dati con altri sistemi  
• Può essere eseguito su contenitori di applicazioni Web come Tomcat, JBoss e così via  
• Può essere ridimensionato verticalmente aumentando la potenza delle macchine su cui viene eseguito o ridimensionato orizzontalmente aggiungendo istanze aggiuntive dietro i bilanciatori del carico

I vantaggi di un'architettura per applicazioni web monolitiche sono:

• Più semplice da sviluppare poiché lo stack tecnologico è uniforme su tutti i livelli.  
• Più semplice da testare in quanto l'intera applicazione è raggruppata in un unico pacchetto, rendendo più semplice l'esecuzione di test di integrazione e end-to-end.  
• Più semplice e veloce da implementare, dato che hai solo un pacchetto di cui preoccuparti.  
• Più semplice da ridimensionare in quanto è possibile moltiplicare il numero di istanze dietro un bilanciamento del carico per ridimensionare.  
• Richiede un team più piccolo per mantenere l'applicazione.  
• I membri del team condividono più o meno lo stesso set di abilità.  
• Lo stack tecnico è più semplice e il più delle volte più facile da imparare.  
• Lo sviluppo iniziale è più rapido e quindi il time-to-market è più veloce.  
• Richiede un'infrastruttura più semplice. Anche un semplice contenitore di applicazioni o JVM sarà sufficiente per eseguire l'applicazione.  
Gli svantaggi di un'architettura per applicazioni web monolitiche sono:

I componenti sono strettamente accoppiati tra loro dando luogo a effetti collaterali indesiderati come le modifiche a un componente che causano una regressione in un altro e così via.  
• Diventa complesso ed enorme nel tempo con conseguente rallentamento dello sviluppo. Le nuove funzionalità impiegheranno più tempo per sviluppare e il refactoring delle funzionalità esistenti sarà più difficile a causa dell'accoppiamento stretto.  
• È necessario ridistribuire l'intera applicazione per eventuali modifiche.  
• È meno affidabile grazie ai moduli strettamente accoppiati. Un piccolo problema in un servizio potrebbe interrompere l'intera applicazione.  
• L'adozione di nuove tecnologie è difficile in quanto è necessario migrare l'intera applicazione. La migrazione incrementale non è possibile la maggior parte del tempo. Quindi molte applicazioni monolitiche finiscono per avere uno stack tecnologico obsoleto.  
• I servizi critici non possono essere ridimensionati singolarmente, con conseguente aumento dell'utilizzo delle risorse poiché è necessario ridimensionare l'intera applicazione.  
• Le enormi applicazioni monolite avranno un tempo di avvio e un utilizzo delle risorse più elevati in termini di CPU e memoria.  
• I tems saranno più interdipendenti e sarà difficile scalarli

***Architettura di microservizi***

L'architettura Microservice può offrire quasi tutte le funzionalità di quella monolitica che abbiamo visto nella sezione precedente. Inoltre, offre molte più funzionalità e flessibilità, e quindi è spesso considerata una scelta superiore per applicazioni complesse. A differenza dell'architettura monolitica, è piuttosto difficile generalizzare l'architettura dei microservizi in quanto potrebbe variare notevolmente a seconda del caso d'uso e dell'implementazione. Ma condividono alcuni tratti comuni e sono, in generale, i seguenti:

I componenti del microservizio sono accoppiati liberamente. I componenti possono essere sviluppati, testati, distribuiti e ridimensionati in modo indipendente senza interrompere altri componenti.  
• I componenti non devono essere sviluppati utilizzando lo stesso stack tecnologico. Ciò significa che un singolo componente può scegliere il proprio stack tecnologico e il proprio linguaggio di programmazione.  
• Utilizzano spesso funzionalità avanzate come rilevamento dei servizi, interruzione dei circuiti, bilanciamento del carico e così via.  
• I componenti di Microservice sono per lo più leggeri e svolgono una funzionalità specifica. Ad esempio, un servizio di autenticazione si preoccuperà solo di autenticare un utente nel sistema.  
• Spesso ha una vasta installazione di monitoraggio e risoluzione dei problemi.

I vantaggi di un'architettura per applicazioni Web di microservizio sono:

• Componenti ad accoppiamento lento che garantiscono un migliore isolamento, più facile da testare e più veloce all'avvio.  
• Turnaround di sviluppo più rapido e tempi di commercializzazione migliori. Le nuove funzionalità possono essere sviluppate più velocemente e le funzionalità esistenti possono essere facilmente rifatte.  
• I servizi possono essere implementati in modo indipendente rendendo l'applicazione più affidabile e facilitando le patch.  
• I problemi, ad esempio una perdita di memoria in uno dei servizi, sono isolati e quindi non faranno cadere l'intera applicazione.  
• L'adozione della tecnologia è più semplice, i componenti possono essere aggiornati indipendentemente nella migrazione incrementale, rendendo possibile avere uno stack diverso per ciascun componente.  
• È possibile stabilire modelli di ridimensionamento più complessi ed efficienti. I servizi critici possono essere ridimensionati in modo più efficace. L'infrastruttura è utilizzata in modo più efficiente.  
• I singoli componenti si avvieranno più velocemente rendendo possibile parallelizzare e migliorare l'avviamento generale.  
• Le squadre saranno meno dipendenti l'una dall'altra. Più adatto per squadre agili.  
Gli svantaggi di un'architettura di applicazioni Web di microservizi sono:

• Più complesso in termini di stack complessivo, poiché i diversi componenti potrebbero avere stack tecnologici diversi che costringono il team a investire più tempo nel tenere il passo con loro.  
• Difficile eseguire test end-to-end e test di integrazione in quanto vi sono più parti mobili nello stack.  
• L'intera applicazione è più complessa da implementare in quanto sono presenti complessità con contenitori e virtualizzazione.  
• Lo scaling è più efficiente ma l'upscaling delle impostazioni è più complesso in quanto richiederebbe funzionalità avanzate come discovery dei servizi, routing DNS e così via.  
• Richiede un team più grande per mantenere l'applicazione in quanto vi sono più componenti e più tecnologie coinvolte.  
• I membri del team condividono diversi set di competenze in base al componente su cui lavorano, rendendo più difficile la sostituzione e la condivisione delle conoscenze.  
• Lo stack tecnico è complesso e il più delle volte più difficile da imparare.  
• Il tempo di sviluppo iniziale sarà più alto, rendendo più lento il time-to-market.  
• Richiede un'infrastruttura complessa. Molto spesso saranno necessari contenitori (Docker) e più contenitori JVM o app per l'esecuzione.

Quando scegliere un'architettura monolitica:

Quando l'ambito dell'applicazione è piccolo e ben definito, e si è sicuri che l'applicazione non crescerà in modo tremendo in termini di funzionalità. Ad esempio, un blog, un semplice sito di acquisti online, una semplice applicazione CRUD e così via.  
• Quando le dimensioni della squadra sono piccole, diciamo meno di otto persone (non è un limite difficile ma piuttosto pratico).  
• Quando il set di abilità medio della squadra è novizio o intermedio.  
• Quando il time-to-market è fondamentale.  
• Quando non si vuole spendere troppo per infrastrutture, monitoraggio e così via.  
• Quando la tua base di utenti è piuttosto piccola e non ti aspetti che cresca. Ad esempio, un'app aziendale che si rivolge a un gruppo specifico di utenti.

***Quando scegliere un'architettura di microservizi:***

• Quando l'ambito dell'applicazione è ampio e ben definito e si è sicuri che l'applicazione crescerà enormemente in termini di funzionalità. Ad esempio, un negozio di e-commerce online, un servizio di social media, un servizio di streaming video con una vasta base di utenti, un provider di API e così via.  
• Quando la dimensione della squadra è grande, ci devono essere abbastanza membri per sviluppare in modo efficace i singoli componenti in modo indipendente.  
• Quando il set di abilità medio della squadra è buono e i membri del team sono fiduciosi sui modelli avanzati di microservizio.  
• Quando il time to market non è critico. L'architettura dei microservizi richiederà più tempo per essere subito in primo piano.  
• Quando si è pronti a spendere di più in infrastrutture, monitoraggio e così via, al fine di migliorare la qualità del prodotto.  
• Quando la tua base di utenti è enorme e ti aspetti che cresca. Ad esempio, un'applicazione di social media destinata agli utenti di tutto il mondo.

***JHipster***

JHipster è un generatore Yeoman che crea applicazioni Spring Boot e Angular / React. È in grado di creare un'architettura monolitica e un'architettura di microservizi con tutte le funzionalità che funzionano immediatamente.

Netflix OSS (https://netflix.github.io) è una raccolta di strumenti open source e software prodotti dal team di NETFLIX, INC orientato verso l'architettura dei microservizi. Lo stack elastico (https://www.elastic.co/products) (precedentemente noto come stack ELK) è una raccolta di strumenti software che aiutano il monitoraggio e l'analisi dei microservizi sviluppati da Elasticsearch (https://www.elastic.co) squadra.

Hipster supporta un numero incredibile di moderne tecnologie di applicazione Web fuori dagli schemi.

Vediamo in breve le diverse tecnologie supportate principalmente per applicazioni monolitiche:

* Tecnologie lato client
* Tecnologie lato server
* Opzioni del database

*Tecnologie lato client*

Il ruolo delle tecnologie lato client nello sviluppo full stack è cresciuto dall'uso di JavaScript per le convalide lato client, fino alla scrittura di applicazioni a pagina intera complete utilizzando i framework MVVM lato client. I framework e le toolchain utilizzate sono diventati complessi e travolgenti per gli sviluppatori che sono nuovi nel panorama lato client. Fortunatamente per noi, JHipster fornisce supporto per la maggior parte delle seguenti tecnologie ampiamente utilizzate, lato client.

HTML5  
HTML5 (https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5) è l'ultimo standard HTML (HyperText Markup Language), che introduce nuovi elementi, attributi e comportamenti. Il termine è usato per riferirsi collettivamente a tutte le tecnologie HTML utilizzate per costruire moderne applicazioni web. Questa iterazione ha introdotto il supporto per funzionalità quali archiviazione offline, WebSockets, web worker, WebGL e altro. JHipster utilizza anche le migliori pratiche dal HTML5 Boilerplate (https://html5boilerplate.com).  
HTML5 Boilerplate è una raccolta di tecnologie moderne, impostazioni predefinite e best practice che consentono di avviare rapidamente lo sviluppo web moderno.

CSS3  
CSS3 (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3) è l'ultima delle specifiche CSS (Cascading Style Sheets). Aggiunge supporto per query multimediali, animazioni, flexbox, angoli arrotondati e molto altro. Il CSS3 rende possibile animare in modo nativo elementi, applicare effetti speciali, applicare filtri e così via per sbarazzarsi dei molti hack JavaScript che sono stati usati in precedenza.  
Flexible Box, o flexbox, è una modalità di layout (https://developer.mozilla.org/en-US/docs/Web/CSS/Layout\_mode) che può essere utilizzata al posto del modello di box utilizzato tradizionalmente. Ciò consente di avere un modello di box flessibile che rende i layout reattivi più facili da gestire senza problemi di float e di compressione dei margini.

Sass

I fogli di stile sintatticamente fantastici (Sass) (http://sass-lang.com) sono un linguaggio di estensione CSS. È preelaborato e convertito in CSS durante la compilazione. Ha una semantica simile ai CSS ed è compatibile al 100% con tutte le versioni di CSS. Supporta inoltre funzioni avanzate come la sintassi nidificata, le variabili, i mixin, l'ereditarietà, i partial e così via. Sass rende possibile riutilizzare i CSS e scrivere fogli di stile manutenibili.

Bootstrap

Bootstrap (https://getbootstrap.com) è un framework di interfaccia utente reattivo per lo sviluppo moderno del web. Offre un approccio mobile-first per lo sviluppo web con utility e componenti dell'interfaccia utente pienamente responsive. Bootstrap 4 è l'ultima versione, utilizza flexbox per il layout ed è completamente scritto in Sass, il che rende più facile la personalizzazione. Bootstrap supporta un framework di griglia a 12 colonne, che consente di creare facilmente pagine Web reattive. JHipster utilizza ng-bootstrap (https://ng-bootstrap.github.io) in modo che vengano utilizzati componenti Angular puri invece di quelli forniti da Bootstrap, che vengono creati utilizzando JQuery e Bootstrap viene utilizzato solo per lo stile.  
Il primo sviluppo web mobile è un approccio in cui UX / UI è progettato per schermi di dimensioni più piccole, quindi ti costringe a concentrarti sui dati / elementi più importanti da presentare. Questo design viene poi gradualmente migliorato per schermi di dimensioni maggiori, rendendo il risultato finale reattivo ed efficiente.

MVVM framework

Model-View-View-Model (MVVM) è un modello architettonico sviluppato originariamente da Microsoft. Aiuta ad astrarre o separare lo sviluppo lato client (GUI) dal lato server (modello dati). Il modello di vista è un'astrazione della vista e rappresenta lo stato dei dati nel modello. Con JHipster, puoi scegliere tra Angolare e Reagire come framework lato client.

Angular

AngularJS (https://angularjs.org) (versione 1.x) è un framework MVVM lato client, gestito da Google, che aiuta a sviluppare Single Page Applications (SPA). Si basa su un modello di programmazione dichiarativa e estende HTML standard con la possibilità di aggiungere ulteriori comportamenti, elementi e attributi attraverso direttive.

Angular (https://angular.io) (versione 2 e successive) è una completa riscrittura del framework e quindi non è retrocompatibile con AngularJS. Angular è scritto in TypeScript e raccomanda l'uso di TypeScript per scrivere anche applicazioni angolari. Angular ha rimosso alcuni dei concetti utilizzati in AngularJS come scope, controller, factory e così via. Ha anche una sintassi diversa per il binding di attributi ed eventi. Un'altra importante differenza è che la libreria Angular è modulare e quindi è possibile scegliere i moduli necessari per ridurre le dimensioni del fascio. Angular ha anche introdotto concetti avanzati come AOT (Ahead of Time Compilation), caricamento lento, programmazione reattiva e così via.  
TypeScript è un superset di ECMAScript 6 (ES6 - versione 6 di JavaScript) ed è retrocompatibile con ES5. Ha funzionalità aggiuntive come la tipizzazione statica, i generici, i modificatori di visibilità degli attributi di classe e così via. Poiché TypeScript è un superset di ES6, possiamo anche utilizzare le funzionalità di ES6 (http://es6-features.org) come moduli, lambdas (funzioni di freccia), generatori, iteratori, modelli di stringhe, riflessione, operatori di spread e così via .

React

React (https://reactjs.org) non è un framework MVVM completo. È una libreria JavaScript per la creazione di viste lato client o interfacce utente. È sviluppato e supportato da Facebook e ha una vibrante comunità ed ecosistema alle spalle. React segue un HTML nell'approccio JS e ha un formato speciale chiamato JSX per aiutarci a scrivere i componenti React. A differenza di Angular, React non ha troppi concetti o API da apprendere e quindi è più facile iniziare, ma a React interessa solo il rendering dell'interfaccia utente e quindi di ottenere funzionalità simili offerte da Angular, dovremmo accoppiare React con altre librerie come React Router (https://reacttraining.com/react-router), Redux (https://redux.js.org), MobX (https://mobx.js.org) e così via. JHipster usa React insieme a Redux e React Router e similmente ad Angular, JHipster usa anche TypeScript per React. Ma questo è opzionale in quanto React può essere scritto anche usando JavaScript, preferibilmente ES6 (http://es6-features.org). React è veloce da renderizzare grazie al suo uso di un DOM virtuale (https://reactjs.org/docs/faq-internals.html) per manipolare una vista invece di usare il DOM del browser.  
Se stai iniziando un nuovo progetto, è meglio scegliere Angular o React perché sono ben mantenuti. Tuttavia, con le versioni precedenti di JHipster, AngularJS 1.x è stato offerto anche come opzione, ma sta diventando obsoleto e sarà presto sospeso in JHipster 5.x. JHipster fornirà un modello ufficiale per coloro che sono ancora interessati ad utilizzare AngularJS 1.x. Basta usare il comando jhipster --blueprint generator-jhipster-angularjs per usarlo.

Build tools

Il lato client si è evoluto molto e diventa complesso come il lato server, quindi richiede molti più strumenti nel tuo toolbelt per produrre risultati ottimizzati. Avresti bisogno di uno strumento di compilazione per trasporre, minimizzare e ottimizzare il codice HTML, JavaScript e CSS. Uno dei più popolari è Webpack. JHipster utilizza Webpack per Angular e React.  
Webpack  
Webpack (https://webpack.js.org) è un bundle di moduli con un sistema caricatore / plugin molto flessibile. Webpack percorre il grafico delle dipendenze e lo passa attraverso i caricatori e i plugin configurati. Con Webpack, puoi trascrivere TypeScript in JavaScript, minimizzare e ottimizzare CSS e JS, compilare Sass, revisionare, cancellare le tue risorse e così via. Webpack può rimuovere il codice morto in un processo chiamato tree shaking, riducendo così le dimensioni del bundle. Webpack è configurato utilizzando un file di configurazione e può essere eseguito dalla riga di comando o tramite gli script NPM / YARN.

rowserSync  
BrowserSync (https://browsersync.io) è uno strumento NodeJS che aiuta a testare il browser sincronizzando le modifiche e le interazioni dei file della pagina web su più browser e dispositivi. Fornisce funzionalità come il ricaricamento automatico delle modifiche ai file, le interazioni dell'interfaccia utente sincronizzate, lo scorrimento e così via. Si integra con Webpack / GulpJS per fornire una configurazione di sviluppo produttiva. Semplifica il test di una pagina Web su più browser e dispositivi.

Strumenti di test

Esistono molti framework e strumenti disponibili per il test delle unità, i test end-to-end e così via. JHipster crea test unitari per il codice lato client utilizzando Karma e Jasmine out-of-the-box e supporta anche la creazione di test end-to-end utilizzando Protractor.

Karma  
Karma (https://karma-runner.github.io/2.0/index.html) è un test runner in grado di eseguire codice JavaScript nei browser reali. Crea un server web ed esegue il codice di test contro il codice sorgente. Karma supporta molteplici framework di test come Jasmine, Mocha e Qunit e si integra bene con gli strumenti di integrazione continua.  
Goniometro  
Il goniometro (http://www.protractortest.org) è un framework di test end-to-end sviluppato dal team di Angular. Originariamente era destinato alle applicazioni Angular e AngularJS, ma è abbastanza flessibile da poter essere utilizzato con qualsiasi framework, come React, JQuery, VueJS e così via. Il goniometro esegue test e2e contro i browser reali utilizzando l'API del driver web Selenium.

Internationalization

'internazionalizzazione (i18n) è una funzionalità molto importante in questi giorni e JHipster supporta questo out-of-the-box. Più lingue possono essere scelte durante la creazione dell'applicazione. Sul lato client, ciò si ottiene memorizzando il testo della GUI nei file JSON per lingua e utilizzando una libreria Angular / React per caricarlo dinamicamente in base alla lingua selezionata al runtime.  
Sai perché l'internazionalizzazione è abbreviata in i18n? Perché ci sono 18 caratteri tra I e N. Ci sono altre abbreviazioni con nomi simili nella tecnologia web, ad esempio Accessibilità (a11y), Localizzazione (l10n), Globalizzazione (g11n) e Localizzazione (l12y).

*Tecnologie lato server*

Le tecnologie lato server nello sviluppo web si sono evolute molto e, con l'avvento di framework come Spring e Play, la necessità di Java EE si è ridotta e ha aperto le porte a alternative più ricche di funzionalità, come ad esempio Spring Boot. Alcune delle tecnologie di base come Hibernate sono qui per rimanere, mentre i nuovi concetti come JWT, Liquibase, Swagger, Kafka e WebSockets offrono molte ulteriori opportunità. Diamo un'occhiata ad alcune delle importanti tecnologie supportate da JHipster; li incontreremo più avanti nel libro e daremo un'occhiata più approfondita ad alcune di queste tecnologie.

Spring Framework

The Spring Framework (https://spring.io) potrebbe essere la cosa migliore dal pane affettato nel mondo Java. Ha cambiato il panorama delle applicazioni web Java per il bene. Il panorama è stato monopolizzato dai venditori di JavaEE prima dell'ascesa di Spring e poco dopo la primavera, è diventato la scelta numero uno per gli sviluppatori di Java, dando a JavaEE una corsa per i suoi soldi. Al suo interno, Spring è un contenitore Inversion of Control (IoC) (https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans) che fornisce un'integrazione delle dipendenze e un contesto applicativo. Le caratteristiche principali del triangolo Spring o Spring, combinano IoC, Aspect-Oriented Programming (AOP) (https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#aop), e astrazioni tecnologiche insieme in modo coerente. Il framework ha numerosi moduli destinati a diversi compiti, come gestione dei dati, sicurezza, REST, servizi Web e così via. Spring Framework e i suoi moduli sono gratuiti e open source. Vediamo alcuni dei moduli importanti in modo un po 'più dettagliato.  
IoC è un modello di progettazione software in cui il codice personalizzato o specifico dell'attività è richiamato da una libreria, piuttosto che l'approccio di programmazione procedurale tradizionale in cui il codice personalizzato chiama le librerie quando richiesto. IoC aiuta a rendere il codice più modulare ed estensibile. AOP offre un altro modo di pensare alla struttura del programma. L'unità di modularità è l'aspetto che consente la modularizzazione di problemi come la gestione delle transazioni che taglia trasversalmente più tipi e oggetti.

Spring Boot  
Spring Boot (https://projects.spring.io/spring-boot) è una soluzione ampiamente utilizzata in questi giorni per lo sviluppo di applicazioni Web Java. Ha una convenzione supponente sull'approccio alla configurazione. È completamente guidato dalla configurazione e rende piacevole l'utilizzo di Spring Framework e di molte altre librerie di terze parti. Le applicazioni Spring Boot sono di produzione di qualità e possono essere eseguite in qualsiasi ambiente in cui sia installata una JVM. Utilizza un contenitore di servlet incorporato come Tomcat, Jetty o Undertow per eseguire l'applicazione. Configura automaticamente Spring ovunque sia possibile e dispone di POM di avvio per molti moduli e librerie di terze parti. Non richiede alcuna configurazione XML e consente di personalizzare i bean autoconfigurati utilizzando la configurazione Java.  
JHipster per impostazione predefinita utilizza Undertow come server incorporato nelle applicazioni generate. Undertow è molto leggero e più veloce da avviare ed è ideale per lo sviluppo e la produzione di applicazioni leggere.

Spring Security

Spring Security (https://projects.spring.io/spring-security) è la soluzione di fatto per la sicurezza in un'applicazione basata su Spring Framework. Fornisce API e utility per gestire tutti gli aspetti della sicurezza, come l'autenticazione e l'autorizzazione. Supporta un'ampia gamma di meccanismi di autenticazione come OAuth2, JWT, Session (Web form), LDAP, SSO (Single Sign-On), JAAS (Java Authentication and Authorization Service), Kerberos e così via. Ha anche funzioni come ricordami, sessione simultanea e così via.  
Spring MVC  
Spring MVC (https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html) è la soluzione predefinita per lavorare con l'API Servlet all'interno delle applicazioni Spring. Si tratta di un sistema basato su richiesta e astrae l'API Servlet per semplificare la progettazione dei controller per soddisfare le richieste HTTP. REST è lo standard di fatto per la progettazione di endpoint API in questi giorni e Spring MVC REST è un sottoinsieme specifico che semplifica la progettazione e l'implementazione di servizi RESTful.

Spring data

Spring data

(http://projects.spring.io/spring-data) sono un modulo che astrae le operazioni di accesso ai dati per molte tecnologie e database di accesso ai dati differenti. Fornisce un'API coerente per lavorare senza problemi con diverse implementazioni sottostanti. Questo ci libera dalla preoccupazione per il database sottostante e la tecnologia di accesso ai dati. Ha potenti funzionalità come la generazione di query dinamiche da nomi di metodi, astrazioni di mapping di oggetti personalizzati e così via. I dati Spring supportano il lavoro con JPA, MongoDB, Redis ed Elasticsearch per nominarne alcuni. Permette anche di esportare archivi di dati Spring come risorse RESTful.  
Sicurezza  
Nelle moderne applicazioni Web, esistono diversi modi per implementare l'autenticazione e l'autorizzazione. La sicurezza Spring supporta un'ampia gamma di meccanismi, come abbiamo visto in precedenza, e JHipster fornisce supporto per i seguenti standard.  
JWT  
JSON Web Token (JWT) (https://jwt.io) è uno standard di settore aperto per i token di sicurezza. L'autenticazione JWT funziona da server e client passando e verificando le attestazioni. Un server genera un token JWT e lo restituisce al client quando le credenziali dell'utente vengono convalidate correttamente. Il client memorizzerà questo token localmente e lo userà per richiedere le risorse di protezione dal server in seguito passando il token nell'intestazione della richiesta. Questo è un meccanismo di autenticazione stateless. Questo è spiegato in dettaglio nel capitolo 9, Building Microservices with JHipster.  
Sessione  
L'autenticazione basata sulla sessione è il meccanismo di autenticazione basato sul modulo Web tradizionale in cui il server crea e mantiene una sessione per le credenziali dell'utente convalidate. Questo è stato e normalmente non è molto scalabile a meno che non si utilizzi una sessione HTTP distribuita, che è possibile utilizzando una cache distribuita come Hazelcast o utilizzando le funzionalità di replica di sessione di un server Web dedicato o di un servizio di bilanciamento del carico. JHipster aggiunge molte funzionalità al di sopra del meccanismo standard, come i token protetti che sono archiviati nel DB, e può essere invalidato, utilizzato nel ricordare i meccanismi di me, e così via.  
OAuth2  
OAuth2 (https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth) è un protocollo per l'autenticazione e l'autorizzazione stateless. Il protocollo consente alle applicazioni di ottenere un accesso limitato agli account utente sui servizi. L'autenticazione dell'utente è delegata a un servizio, in genere un server OAuth2. OAuth2 è più complicato da configurare rispetto ai meccanismi menzionati in precedenza. JHipster supporta la configurazione di OAuth con OpenID Connect (OIDC) e può utilizzare Keycloak (https://keycloak.org) o Okta (https://developer.okta.com/blog/2017/10/20/oidc-with-jhipster ) fuori dalla scatola.

Build tools

JHipster supporta l'utilizzo di Maven o Gradle come strumento di compilazione per il codice lato server. Entrambi sono gratuiti e open source.

Maven  
Maven (https://maven.apache.org) è uno strumento di automazione della build che utilizza un documento XML chiamato pom.xml per specificare come viene costruita un'applicazione e le sue dipendenze. I plugin e le dipendenze vengono scaricati da un server centrale e memorizzati nella cache localmente. Il file di build Maven è chiamato un Project Object Model (POM) e descrive il processo di compilazione stesso. Maven ha una lunga storia ed è molto più stabile e affidabile rispetto a Gradle. Ha anche un enorme ecosistema di plugin.  
Gradle  
Gradle (https://gradle.org) è uno strumento di automazione della build che utilizza un DSL Groovy per specificare il piano di build e le dipendenze. È un forte concorrente che sta rapidamente guadagnando popolarità e adozione. Gradle è molto più flessibile e ricco di funzionalità rispetto a Maven, rendendolo la scelta ideale per configurazioni di build molto complesse. L'ultima versione di Gradle supera facilmente Maven in termini di velocità e funzionalità. Un altro vantaggio unico di Gradle è la possibilità di scrivere il codice Groovy standard nello script di build, rendendo possibile eseguire praticamente tutto a livello di programmazione. Ha anche un ottimo supporto per i plugin.  
ibernare  
Hibernate (http://hibernate.org) è lo strumento ORM (Object Relational Mapping) più popolare per Java. Aiuta a mappare un modello di dominio orientato agli oggetti a uno schema di database relazionale utilizzando le annotazioni Java. Implementa JPA (Java Persistence API) ed è il fornitore di riferimento per un'implementazione JPA. Hibernate offre anche molte funzionalità aggiuntive come il controllo delle entità, la convalida dei bean e così via. Hibernate genera automaticamente query SQL a seconda della semantica del database sottostante e consente di cambiare facilmente i database di un'applicazione. Rende inoltre indipendente il database dell'applicazione senza alcun blocco del fornitore. Hibernate è un software gratuito e open source.  
Liquibase  
Liquibase (http://www.liquibase.org) è uno strumento di controllo delle versioni gratuito e open source per il database. Ti consente di tracciare, gestire e applicare le modifiche dello schema del database utilizzando i file di configurazione senza dover interferire con SQL. È indipendente dal database e si integra perfettamente con JPA, rendendo il database dell'applicazione indipendente. Liquibase può essere eseguito dall'interno dell'applicazione, semplificando l'impostazione e la gestione del database ed eliminando la necessità di un DBA per la maggior parte della gestione dei DB. Liquibase può anche aggiungere / rimuovere dati da / verso un database, rendendolo utile anche per le migrazioni.  
caching  
Il caching è una buona pratica nello sviluppo del software e migliora notevolmente le prestazioni delle operazioni di lettura. La memorizzazione nella cache può essere abilitata per la cache di secondo livello di Hibernate e anche con l'astrazione di Spring Cache per abilitare il caching a livello di metodo. JHipster supporta la cache di secondo livello di Hibernate compatibile con JCache fornita da EhCache, Hazelcast e Infinispan.  
EHCache  
Ehcache (http://www.ehcache.org) è un fornitore JCache open source ed è una delle soluzioni di caching Java più utilizzate. È compatibile con JCache ed è una buona scelta per le applicazioni che non sono raggruppate in cluster. Per gli ambienti cluster, sono necessari server Terracotta aggiuntivi. È stabile, veloce e semplice da configurare.  
Hazelcast  
Hazelcast (https://hazelcast.org) è una soluzione di griglia di dati in-memory distribuita open source. Ha un eccellente supporto per applicazioni cluster e ambienti distribuiti e quindi diventa una buona scelta per il caching. Mentre Hazelcast ha numerose altre funzionalità e casi d'uso, il caching rimane uno dei più importanti. È altamente scalabile e una buona opzione per i microservizi grazie alla sua natura distribuita.

nfinispan  
Infinispan (http://infinispan.org) è una cache distribuita e un archivio di valori-chiave di Red Hat. È gratuito e open source. Supporta ambienti cluster ed è quindi una buona scelta per i microservizi. Ha più funzioni come griglie di dati in memoria, supporto MapReduce e così via.

Swagger

La specifica OpenAPI (precedentemente nota come specifica Swagger) è uno standard aperto per la progettazione e il consumo di API e servizi web RESTful. La specifica OpenAPI è uno standard fondato da una varietà di società tra cui Google, Microsoft e IBM. Il nome Swagger (https://swagger.io) viene ora utilizzato per gli strumenti associati. JHipster supporta il primo modello di sviluppo API con Swagger code-gen e supporta anche la visualizzazione dell'API con l'interfaccia utente Swagger.  
Thymeleaf  
Thymeleaf (http://www.thymeleaf.org) è un motore di template sul lato server Java open source con un'ottima integrazione con Spring. Thymeleaf può essere utilizzato per generare pagine Web sul lato server, per la creazione di modelli di messaggi e-mail e così via. Sebbene i modelli di pagine Web sul lato server stiano lentamente perdendo terreno ai framework MVVM lato client, è comunque uno strumento utile se si desidera avere qualcosa di più di un'applicazione a singola pagina usando Angular.  
Metriche Dropwizard  
Le metriche Dropwizard (http://metrics.dropwizard.io/4.0.0/) sono un'eccellente libreria open source per misurare le prestazioni della tua applicazione web Java. Accoppiato con Spring Boot, questo può portare un sacco di valore misurando le prestazioni dell'API REST, misurando le prestazioni del livello di cache e del database e così via. Dropwizard fornisce annotazioni a portata di mano per contrassegnare i metodi da monitorare. Supporta contatori, timer e così via.  
WebSocket  
WebSocket (https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API) è un protocollo di comunicazione che funziona su TCP. Fornisce un canale di comunicazione full duplex su una singola connessione TCP. È stato standardizzato dal W3C (https://www.w3.org). È leggero e consente la comunicazione in tempo reale tra un client e un server. In termini di applicazioni Web, ciò consente al server di comunicare con l'app client nel browser senza una richiesta da parte del client. Questo apre la porta a spingere i dati dal server al client in tempo reale e per implementazioni come chat in tempo reale, notifiche e così via. Sul lato server, JHipster si affida a Spring, che fornisce il supporto necessario (https://spring.io/guides/gs/messaging-stomp-websocket/) per funzionare con WebSocket.  
Kafka  
Kafka (https://kafka.apache.org) è un sistema di elaborazione del flusso open source. Ha una coda di messaggi distribuita pub / sub-based per l'archiviazione. La tolleranza agli errori e la scalabilità l'hanno aiutato a sostituire JMS e AMQP come la coda di messaggistica preferita. Spring fornisce un'astrazione su Kafka per semplificare la configurazione e l'utilizzo di Kafka.  
JMS (Java Message Service) (https://en.wikipedia.org/wiki/Java\_Message\_Service) è uno standard di messaggistica sviluppato per Java EE e consente di inviare e ricevere messaggi asincroni tra componenti utilizzando argomenti e code. AMQP (Advanced Message Queuing Protocol) (https://www.amqp.org/) è un protocollo standard aperto per middleware orientato ai messaggi, che fornisce funzionalità quali l'accodamento, il routing e i meccanismi di sottoscrizione alla pubblicazione.  
Strutture di prova  
I test sul lato server possono essere principalmente suddivisi in test unitari, test di integrazione, test delle prestazioni e test comportamentali. JHipster supporta tutti questi strumenti con i seguenti strumenti, dai quali JUnit è pronto all'uso e altri sono opt-in.  
JUnit  
JUnit (https://junit.org/junit5/) è il framework di testing Java più utilizzato. È un software gratuito e open source. Inizialmente era previsto per il test unitario, ma combinato con Spring Test Framework (https://docs.spring.io/spring/docs/current/spring-framework-reference/testing.html#testing-introduction) può essere utilizzato anche per Test d'integrazione. JHipster crea test unitari e test di integrazione API REST utilizzando JUnit e Spring Test Framework.  
Gatling  
Gatling (https://gatling.io/) è uno strumento gratuito e open source per test delle prestazioni e del carico. È basato su Scala e utilizza una DSL Scala per scrivere le specifiche del test. Crea report dettagliati sul test del carico e può essere utilizzato per simulare tutti i tipi di carico su un sistema. È uno strumento necessario per le applicazioni critiche per le prestazioni.

Cucumber

Cucumber (https://cucumber.io/) è un framework di testing Behaviour Driven (BDD) utilizzato principalmente per i test di accettazione. Usa un parser di linguaggio chiamato Gherkin, che è leggibile da un punto di vista umano e somiglia molto al semplice inglese.

Introduzione alle opzioni del database  
Oggi, ci sono una grande varietà di opzioni di database là fuori. Questi possono essere ampiamente classificati nel seguente:  
• Database SQL  
• Database NoSQL  
Puoi visitare; https://db-engines.com/en/ranking per vedere la popolarità di diversi database.

JHipster supporta alcuni dei database più diffusi, come descritto qui.  
Database SQL  
Database SQL o RDBMS (Relational Database Management Systems) sono quelli che supportano un modello di dati orientato alla tabella relazionale. Supportano lo schema della tabella definito dal nome fisso e il numero di colonne / attributi con un tipo di dati fisso. Ogni riga in una tabella contiene un valore per ogni colonna. Le tabelle possono essere correlate tra loro.

Introduzione alle opzioni del database  
Oggi, ci sono una grande varietà di opzioni di database là fuori. Questi possono essere ampiamente classificati nel seguente:  
• Database SQL  
• Database NoSQL  
Puoi visitare; https://db-engines.com/en/ranking per vedere la popolarità di diversi database.  
JHipster supporta alcuni dei database più diffusi, come descritto qui.  
Database SQL  
Database SQL o RDBMS (Relational Database Management Systems) sono quelli che supportano un modello di dati orientato alla tabella relazionale. Supportano lo schema della tabella definito dal nome fisso e il numero di colonne / attributi con un tipo di dati fisso. Ogni riga in una tabella contiene un valore per ogni colonna. Le tabelle possono essere correlate tra loro.  
H2  
H2 (http://www.h2database.com/html/main.html) è un RDBMS incorporato gratuito comunemente usato per lo sviluppo e il test. Normalmente può essere eseguito in modalità file system per la persistenza o la modalità in memoria. Ha un ingombro molto ridotto ed è estremamente facile da configurare e utilizzare. Non ha molte delle funzionalità aziendali offerte da altri motori di database tradizionali e quindi di solito non è preferito per l'utilizzo di produzione.  
MySQL  
MySQL (https://www.mysql.com/) è uno dei più popolari motori di database ed è gratuito e open source. È di Oracle ma ha anche una comunità molto vivace. Ha funzionalità pronte per l'uso come sharding, replicazione, partizionamento e così via. Di questi tempi è uno dei database SQL preferiti.  
MariaDB  
MariaDB (https://mariadb.org/) è un motore di database compatibile con MySQL con un'attenzione particolare alla sicurezza, alle prestazioni e all'alta disponibilità. Sta guadagnando popolarità ed è ricercato come una buona alternativa a MySQL. È un software gratuito e open source.  
PostgreSQL  
PostgreSQL (https://www.postgresql.org/) è un altro sistema di database libero e open source che è molto richiesto. È gestito attivamente da una comunità. Una delle caratteristiche uniche di PostgreSQL è l'archiviazione avanzata degli oggetti JSON con la capacità di indicizzare e interrogare all'interno di JSON. Ciò rende possibile utilizzarlo come database NoSQL o in modalità ibrida. Dispone inoltre di funzionalità pronte all'azienda come replica, alta disponibilità e così via.  
MS SQL  
Il server MS SQL (https://www.microsoft.com/nl-nl/sql-server/sql-server-2017) è un sistema di database aziendale sviluppato e supportato da Microsoft. È un software commerciale e richiede una licenza a pagamento da utilizzare. Dispone di funzionalità enterprise-ready e supporto premium di Microsoft. È una delle scelte popolari per i sistemi mission-critical.  
Oracolo  
Oracle (https://www.oracle.com/database/index.html) è il database più utilizzato grazie alle sue funzionalità legacy e aziendali. È un software commerciale e richiede una licenza a pagamento da utilizzare. Ha funzionalità pronte per l'azienda come sharding, replicazione, alta disponibilità e così via.  
Database NoSQL  
Questo è un ampio ambito che comprende qualsiasi database che non sia un RDBMS. Ciò include registrazione di documenti, la memorizzazione di grandi colonne, search engines, memorizzazione di valore-chiave, DBMS grafico, archivi di contenuti e così via. Una caratteristica generale di tali database è che possono essere senza schema e non fare affidamento sui dati relazionali.  
MongoDB  
MongoDB (https://www.mongodb.com/) è un archivio di documenti multipiattaforma ed è una delle scelte più popolari per i database NoSQL. Ha un'API e un linguaggio di query proprietari basati su JSON. Supporta MapReduce e funzionalità aziendali come sharding, replica e così via. È un software gratuito e open source.  
MapReduce è un paradigma di elaborazione dei dati in cui un lavoro è suddiviso in più attività di mappe parallele, con l'output prodotto ordinato e ridotto nel risultato. Ciò rende l'elaborazione di dataset di grandi dimensioni efficiente e più veloce.  
cassandra  
Apache Cassandra (http://cassandra.apache.org/) è fatto di colonne distribuite memorizzate con particolare attenzione alla disponibilità elevata, alla scalabilità e alle prestazioni. A causa della sua natura distribuita, non ha un singolo punto di errore che lo rende la scelta più popolare per i sistemi critici di alta disponibilità. È stato originariamente sviluppato e aperto da Facebook.  
Sapevi che Cassandra può contenere fino a 2 miliardi di colonne per riga?  
elasticsearch  
Elasticsearch (https://www.elastic.co/products/elasticsearch) è un motore di ricerca e analisi basato su Apache Lucene (http://lucene.apache.org/). È tecnicamente un database NoSQL, ma è principalmente utilizzato come motore di ricerca a causa della sua capacità di indicizzazione e prestazioni elevate. Può essere distribuito e multi-tenant con funzionalità di ricerca full-text. Ha un'interfaccia web e documenti JSON. È uno dei motori di ricerca più utilizzati.

***Installazione e configurazione***Per iniziare con JHipster, dovrai installare lo strumento CLI di JHipster. La CLI JHipster viene fornita con i comandi necessari per utilizzare tutte le funzionalità offerte dalla piattaforma.  
JHipster online: se desideri creare un'applicazione senza installare nulla, puoi farlo visitando https://start.jhipster.tech. Puoi autorizzare l'applicazione a generare un progetto direttamente nel tuo account GitHub o puoi scaricare il sorgente come file ZIP.  
Prerequisiti  
Prima di installare JHipster CLI, diamo un'occhiata ai prerequisiti. Dovremo installare alcune dipendenze e configurare il nostro IDE preferito per funzionare al meglio con il codice generato. È possibile visitare http://www.jhipster.tech/installation/ per ottenere informazioni aggiornate su questo.

Installazione e configurazione  
Per iniziare con JHipster, dovrai installare lo strumento CLI di JHipster. La CLI JHipster viene fornita con i comandi necessari per utilizzare tutte le funzionalità offerte dalla piattaforma.  
JHipster online: se desideri creare un'applicazione senza installare nulla, puoi farlo visitando https://start.jhipster.tech. Puoi autorizzare l'applicazione a generare un progetto direttamente nel tuo account GitHub o puoi scaricare il sorgente come file ZIP.  
Prerequisiti  
Prima di installare JHipster CLI, diamo un'occhiata ai prerequisiti. Dovremo installare alcune dipendenze e configurare il nostro IDE preferito per funzionare al meglio con il codice generato. È possibile visitare http://www.jhipster.tech/installation/ per ottenere informazioni aggiornate su questo.  
Strumenti richiesti  
Di seguito sono riportati gli strumenti necessari per installare JHipster e per lavorare con le applicazioni generate. Se non li hai già installati, segui questi passaggi e installali.  
Sarà necessario utilizzare un'interfaccia della riga di comando (prompt dei comandi o applicazione terminale) in questa sezione e quindi è meglio averne una aperta. Poiché l'installazione di alcuni dei seguenti strumenti altera le variabili di ambiente, potrebbe essere necessario chiudere e riaprire il Terminale dopo l'installazione di uno strumento:  
• Su Windows, utilizzare il prompt dei comandi predefinito (CMD) o PowerShell  
• Su Linux, usa Bash o il tuo emulatore di terminale preferito  
• Su macOS, usa iTerm o la tua applicazione Terminale preferita  
Procedura d'installazione  
Vediamo la procedura di installazione per ciascuno degli strumenti.  
Java 8  
Java 9 è l'ultima versione di Java che introduce funzionalità come moduli, flussi reattivi e così via. Mentre le applicazioni JHipster funzionano con Java 9, si consiglia di attenersi al più stabile Java 8 fino a quando il supporto di Java 9 non è stabile in tutte le dipendenze utilizzate.  
Le applicazioni generate utilizzano Java 8 e quindi è necessario compilare le applicazioni:  
1. Verifica la tua versione di Java installata eseguendo il comando java -version nel Terminale. Dovrebbe visualizzare la versione java "1.8.x" dove x potrebbe essere qualsiasi versione di patch.  
2. Se non è installata la versione corretta, visitare il sito Web Oracle (http://www.oracle.com/technetwork/java/javase/downloads/index.html) e seguire le istruzioni per installare JDK per Java 8.  
3. Una volta installato, controlla di nuovo il comando nel passaggio 1 per essere sicuro. Poiché il JDK altera la variabile di ambiente per impostare JAVA\_HOME, devi aprire un nuovo Terminale qui.

Git

Git è il sistema di controllo delle versioni più utilizzato per la gestione del codice sorgente. Promuove il controllo di revisione distribuito ed è parte integrante dello sviluppo in questi giorni.  
JHipster utilizza Git per l'aggiornamento delle applicazioni e Git è anche raccomandato per il buon funzionamento degli ecosistemi NodeJS e NPM:  
1. Controlla Git eseguendo git --version nel Terminale. Dovrebbe visualizzare git versione x.x.x; il numero di versione può essere qualsiasi cosa.  
2. Se il comando non viene trovato, è possibile visitare git-scm (https://git-scm.com/downloads) e seguire le istruzioni per installare Git sul proprio sistema operativo.  
3. Una volta installato, controlla di nuovo il comando nel passaggio 1 per essere sicuro.  
Node.js  
Node.js è un ambiente runtime JavaScript. Ha rivoluzionato il mondo JavaScript e reso JavaScript il linguaggio di sviluppo più popolare tra gli sviluppatori oggi (secondo https://insights.stackoverflow.com/survey/2017#technology-programming-languages). L'ecosistema Node è il più grande del mondo con oltre 600.000 pacchetti ed è gestito da NPM, il gestore di pacchetti predefinito.  
La CLI JHipster è un'applicazione NodeJS e quindi richiede l'esecuzione di NodeJS e molti degli strumenti utilizzati nell'applicazione generata richiedono anche NodeJS:  
1. Controlla NodeJS digitando node -v nel Terminale. Dovrebbe visualizzare un numero di versione. Assicurarsi che il numero di versione sia maggiore di 8.9 e corrisponda all'ultima versione LTS di NodeJS.  
2. Se il comando non viene trovato o se si dispone di una versione inferiore di NodeJS, visitare il sito Web Node.js (https://nodejs.org/en/download/) e seguire le istruzioni per installare la versione LTS più recente a disposizione. Si noti che le versioni non LTS (attuali) potrebbero non essere stabili e si consiglia di non utilizzarle.  
3. Una volta installato, controlla di nuovo il comando nel passaggio 1 per essere sicuro. Dato che NodeJS altera le variabili di ambiente, dovresti aprire un nuovo Terminale qui.  
4. NPM viene installato automaticamente quando si installa NodeJS. Puoi verificarlo eseguendo npm -v nel Terminale.  
È possibile installare più pacchetti NPM eseguendo il comando npm -g install bower gulp-cli CLI o usando Yarn, filato globale add bower gulp-cli.

Yarn

Yarn è un gestore di pacchetti per NodeJS. È API e funzionalità compatibili con NPM e offre prestazioni migliori e una struttura piatta dei pacchetti.  
JHipster, per impostazione predefinita, utilizza Filato invece di NPM poiché Yarn è molto più veloce al momento della scrittura. Se si preferisce utilizzare NPM, è possibile saltare questo passaggio:  
1. È possibile visitare il sito Web di Yarn (https://yarnpkg.com/en/docs/install) e seguire le istruzioni per installare Yarn.  
2. Una volta installato, controllare eseguendo il filo --versione per essere sicuri.  
docker  
Docker è lo standard defacto per la gestione dei container e ha reso l'uso dei container un gioco da ragazzi. Fornisce strumenti per creare, condividere e distribuire contenitori.  
 Avrai bisogno di Docker e docker-compose per eseguire le immagini del database generato e per lo sviluppo di microservizi:  
1. Cerca Docker eseguendo la finestra mobile -v in un terminale. Dovrebbe visualizzare un numero di versione.  
2. Verifica la composizione del docker eseguendo la finestra mobile-compose -v in un Terminale. Dovrebbe visualizzare un numero di versione. Se sei su Mac o Linux, puoi semplicemente eseguire docker -v && docker-compose -v insieme.  
3. Se il comando non viene trovato, è possibile visitare il sito Web di Docker (https://docs.docker.com/install/) e seguire le istruzioni per installarlo. Inoltre, installa Docker Compose (https://docs.docker.com/compose/install/) seguendo le istruzioni.  
4. Una volta installato, controlla di nuovo il comando nel passaggio 1 per essere sicuro.  
Opzionalmente Installare uno strumento di creazione Java: Normalmente JHipster installerà automaticamente Maven Wrapper (https://github.com/takari/maven-wrapper) o Gradle Wrapper (https://docs.gradle.org/current/userguide/gradle\_wrapper .html) per te, in base alla scelta dello strumento di creazione. Se non vuoi utilizzare questi wrapper, vai al sito web ufficiale di Maven (http://maven.apache.org/) o al sito web di Gradle (https://gradle.org/) per fare la tua installazione.

***Configurazione IDE***  
Le applicazioni JHipster possono essere create utilizzando un'interfaccia della riga di comando e JHipster CLI. Tecnicamente parlando, un IDE non è un requisito ma quando si continua lo sviluppo di un'applicazione generata si consiglia vivamente di utilizzare un IDE Java appropriato come IntelliJ, Eclipse o Netbeans. A volte potresti anche utilizzare editor di testo avanzati come Visual Studio Code o Atom con plugin appropriati per portare a termine il lavoro. A seconda dell'ID / editor di testo scelto, è consigliabile utilizzare i seguenti plug-in per rendere lo sviluppo più produttivo:

* Angular/React: Tslint, TypeScript, editor config
* Java: Spring, Gradle/Maven, Java Language support (VS Code)

***Indipendentemente dall'editor di testo / IDE, escludere sempre le cartelle node\_modules, git, build e target per accelerare l'indicizzazione.*** Alcuni IDE lo faranno automaticamente in base al file .gitignore.

Visita http://www.jhipster.tech/configuring-ide/ nel tuo browser preferito per saperne di più.  
Configurazione di sistema  
Prima di installare e immergerti in JHipster, ecco alcuni suggerimenti per prepararti ad alcuni dei problemi comuni che potresti incontrare:  
• Quando si utilizza Yarn su macOS o Linux, è necessario avere $HOME/.config/yarn/global/node\_modules/.bin nel percorso.

Normalmente questo verrà fatto automaticamente quando installi il filato, ma in caso contrario, puoi eseguire il comando PATH="$PATH:`yarn global bin`:$HOME/.config/yarn/global/node\_modules/.bin" in un terminale per fare questo.

Se si è dietro un proxy aziendale, sarà necessario ignorarlo per consentire a NPM, Bower e Maven / Gradle di funzionare correttamente. Visita http://www.jhipster.tech/configuring-a-corporate-proxy/ per vedere quali opzioni proxy possono essere impostate per diversi strumenti utilizzati.  
Se utilizzi Mac o Linux e se utilizzi Oh-My-Zsh o la shell Fisherman, puoi utilizzare i plug-in specifici di JHipster per questo. Visita http://www.jhipster.tech/shell-plugins/ per i dettagli.  
Installazione di JHipster  
OK, ora iniziamo per davvero. JHipster può essere utilizzato da un'installazione locale con NPM o Yarn, da un'immagine di Vagrant fornita dal team o utilizzando un'immagine Docker. In alternativa, c'è anche l'applicazione online JHipster che abbiamo visto in precedenza.  
Tra tutte le opzioni, il modo migliore per utilizzare tutta la potenza di JHipster è installare JHipster CLI usando Yarn o NPM. Apri un terminale ed esegui:

> yarn add global generator-jhipster

Se preferisci NPM, esegui semplicemente:

> npm install -g generator-jhipster

Se sei qualcuno che non può aspettare l'arrivo di nuove versioni, puoi sempre utilizzare il codice di sviluppo corrente seguendo questi passaggi dopo aver installato la CLI JHipster seguendo i passaggi precedenti:  
1. In un terminale, accedere a una directory che si desidera utilizzare. Ad esempio, se hai una cartella chiamata project nella tua home directory, esegui cd ~ / projects / e per Windows esegui cd c: \ Users \ <username> \ Desktop \ projects  
2. Esegui, git clone https://github.com/jhipster/generator-jhipster.git  
3. Ora vai alla cartella eseguendo cd generator-jhipster  
4. Eseguire il collegamento npm per creare un collegamento simbolico da questa cartella nell'applicazione installata globalmente in node\_modules globali  
5. Ora, quando si eseguono i comandi JHipster, si utilizzerà la versione clonata anziché la versione installata

*Nota che dovresti farlo solo se sei assolutamente sicuro di ciò che stai facendo.*

***Costruire applicazioni Web monolitiche con JHipster***

Entriamo in azione e costruiamo un'applicazione web di livello produttivo utilizzando JHipster. Prima di iniziare, abbiamo bisogno di un caso d'uso. Creeremo un'applicazione web di e-commerce che gestisce prodotti, clienti, i loro ordini e fatture. L'applicazione web utilizzerà un database MySQL per la produzione e avrà un front-end angolare. L'interfaccia utente per il sito web di acquisto effettivo sarà diversa dalle funzionalità di back office, che saranno disponibili solo per i dipendenti che hanno un ruolo di amministratore. Per questo esercizio, costruiremo solo un'interfaccia utente semplice per la parte rivolta al cliente.

*Generazione di applicazioni*  
Prima di iniziare a generare l'applicazione, dobbiamo preparare il nostro spazio di lavoro poiché questo spazio di lavoro verrà utilizzato in questo libro e durante la nostra creazione creeremo molti rami Git in questo spazio di lavoro.  
Visita http://rogerdudler.github.io/git-guide/ per una guida rapida di riferimento sui comandi Git.  
***Passaggio 1: preparazione dell'area di lavoro***

Creiamo una nuova cartella per lo spazio di lavoro. Crea una cartella chiamata e-commerce-app e dal terminale, vai alla cartella:

> mkdir e-commerce-app

> cd e-commerce-app

Ora crea una nuova cartella per la nostra applicazione; chiamiamolo negozio online e naviga verso di esso:

> mkdir online-store

> cd online-store

Ora siamo pronti per invocare JHipster. Prima assicuriamoci che tutto sia pronto eseguendo il comando jhipster --version. Dovrebbe stampare una versione JHipster installata globalmente, altrimenti dovrai seguire le istruzioni del capitolo precedente per configurarlo.

***Passaggio 2: generazione del codice utilizzando JHipster***

Inizializza JHipster eseguendo il comando jhipster nel terminale, che produrrà il seguente output:



JHipster chiederà un numero di domande per ottenere informazioni sulle diverse opzioni richieste. La prima domanda riguarda il tipo di applicazione che vogliamo e ci vengono presentate le seguenti quattro opzioni:

* Monolithic application: come suggerisce il nome, crea un'applicazione Web monolitica con un back-end basato su Spring Boot e un front-end SPA.
* Microservice application: crea un microservizio Spring Boot senza alcun frontend ed è progettato per funzionare con un'architettura microservice JHipster.
* Microservice gateway: crea un'applicazione Spring Boot molto simile all'applicazione monolitica ma orientata verso un'architettura di microservizi con configurazioni aggiuntive. È dotato di un frontend SPA.
* JHipster UAA server: crea un servizio di autenticazione utente e autorizzazione OAuth2. Questo non avrà alcun codice di frontend ed è progettato per essere utilizzato in un'architettura di microservizi JHipster.

Sceglieremo l'applicazione monolitica per il nostro caso d'uso.

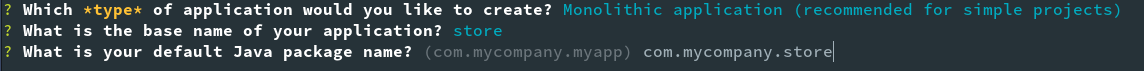
Esegui jhipster --help per vedere tutti i comandi disponibili. Esegui jhipster <comando> --help per vedere le informazioni di aiuto per un comando specifico; ad esempio, l'app jhipster --help mostrerà le informazioni della guida per il processo principale di generazione delle app.

***Opzioni lato server***

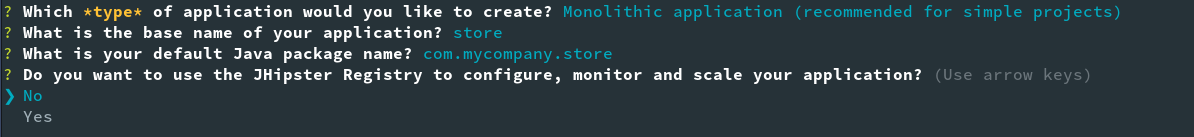
l generatore inizierà ora a chiederci quali siano le opzioni lato server di cui abbiamo bisogno. Passiamo attraverso di loro uno per uno:  
• Domanda 1: questo prompt richiede un nome base per l'applicazione, che viene utilizzato per creare i nomi di file di classe principale, i nomi di database e così via. Per impostazione predefinita, JHipster suggerisce il nome della directory corrente se non contiene caratteri speciali nel nome. Diamo un nome alla nostra applicazione come negozio. Tieni presente che i file verranno creati nella directory corrente in cui ti trovi:



• Domanda 2: questo prompt richiede un nome pacchetto Java. Scegliamo com.mycompany.store:



• Domanda 3. Questo messaggio chiede se è necessario configurare il registro JHipster per questa istanza. Il registro JHipster fornisce un'implementazione del servizio e l'implementazione del server di configurazione che è molto utile per la gestione centralizzata della configurazione e il ridimensionamento dell'applicazione. Per questo caso d'uso, non ne abbiamo bisogno, quindi scegliamo No. Impareremo di più sul Registro JHipster nel Capitolo 8, Introduzione a Microservice Server-Side Technologies, di questo libro:



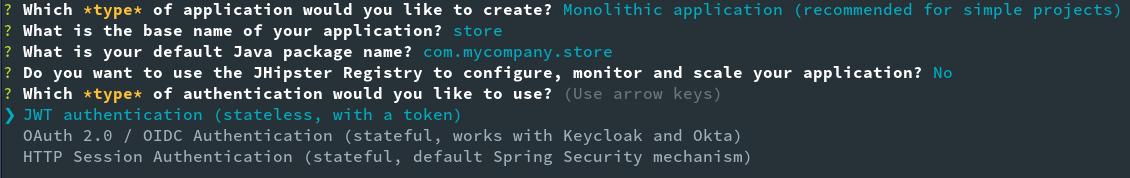
• Domanda 4: questo messaggio richiede di selezionare un meccanismo di autenticazione. Ci vengono presentate tre opzioni:

o JWT authentication

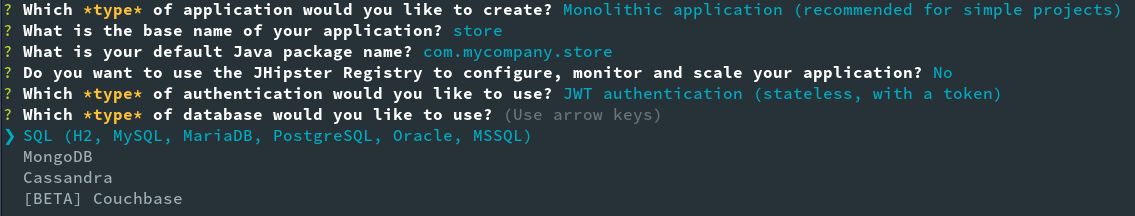
o HTTP Session Authentication

o OAuth 2.0/OIDC Authentication

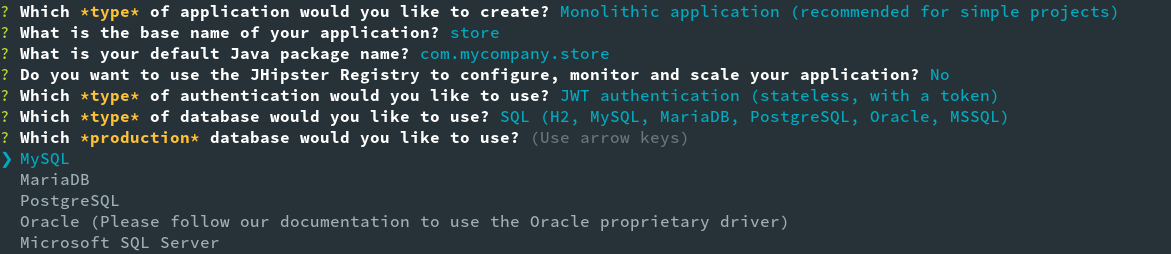
Abbiamo già visto come questi rimandano nel capitolo precedente, e per il nostro caso d'uso, scegliamo l'autenticazione JWT:



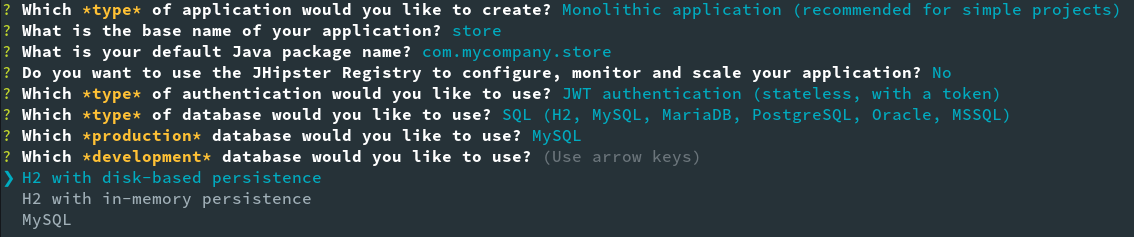
• Domanda 5: questo prompt ci chiede di selezionare un tipo di database; le opzioni fornite sono SQL, MongoDB, Couchbase e Cassandra. Abbiamo già imparato a conoscere le diverse opzioni del database nel capitolo precedente. Per la nostra applicazione, scegliamo un database SQL:



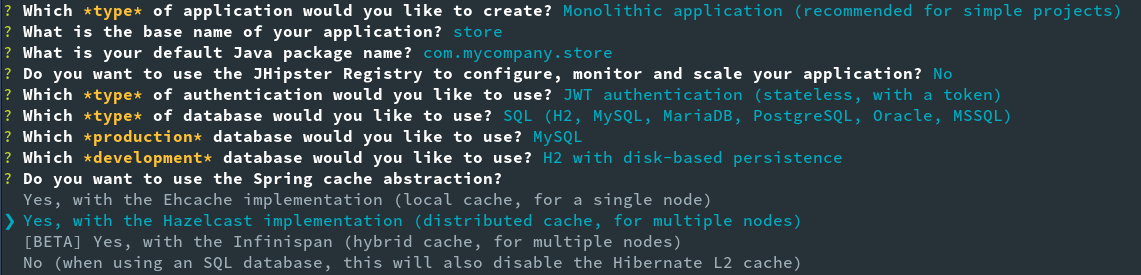
• Domanda 6: questo prompt ci chiede di scegliere un database SQL specifico che vorremmo usare in produzione; le opzioni disponibili sono MySQL, MariaDB, PostgreSQL, Oracle e Microsoft SQL Server. Scegliamo MySQL qui:



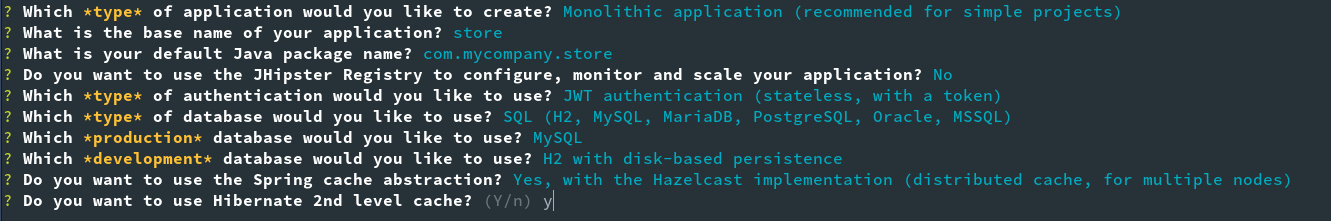
• Domanda 7: questo prompt ci chiede di scegliere tra il nostro database SQL scelto e il database H2 incorporato per lo sviluppo. H2 embedded DB è particolarmente utile in quanto rende lo sviluppo più veloce e autonomo, senza la necessità di avere un'istanza MySQL in esecuzione. Quindi, scegliamo la persistenza basata su disco H2 in quanto è leggera e più facile nello sviluppo rispetto ad avere un servizio DB a pieno titolo in esecuzione:



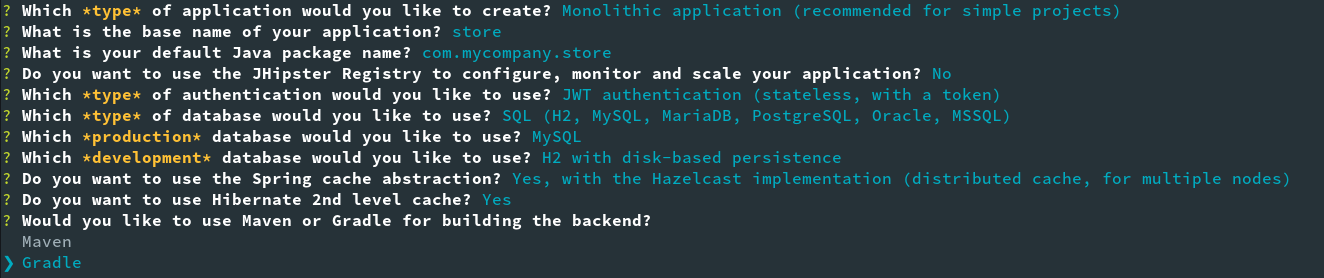
Se il tuo caso d'uso richiede di lavorare con dati persistenti in fase di sviluppo e se il modello non cambierà spesso, potresti anche scegliere MySQL per lo sviluppo in quanto ti darebbe un tempo di avvio più veloce. Questo perché il DB H2 incorporato non ha bisogno di essere inizializzato, ma lo svantaggio è ogni volta che si apportano modifiche allo schema o si ricreano entità, si dovrebbe aggiornare manualmente il DB utilizzando i changelog di liquibase generati, oppure pulire manualmente il DB e iniziare di nuovo Con un DB H2 incorporato, è possibile eseguire ./gradlew clean per cancellarlo.  
• Domanda 8: questo prompt ci chiede di scegliere un'implementazione della cache di primavera. Abbiamo la possibilità di scegliere tra nessuna cache, EHCache, Hazelcast e Infinispan. Da quando abbiamo saputo di questi nel capitolo precedente, andiamo avanti e scegliamo Hazelcast qui:



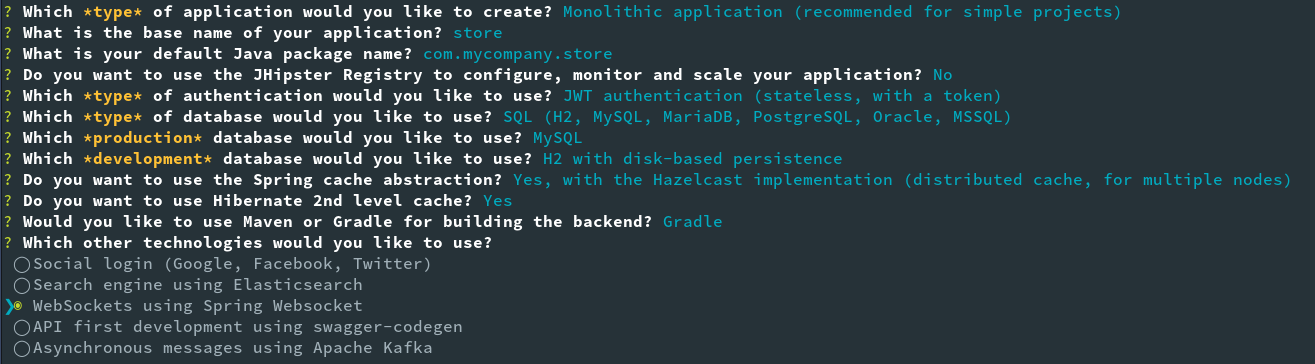
• Domanda 9. Questo prompt ci chiede di scegliere se abbiamo bisogno di una cache di secondo livello per Hibernate. Scegliamo Sì. Utilizzerà la stessa implementazione cache che abbiamo scelto per la domanda precedente:



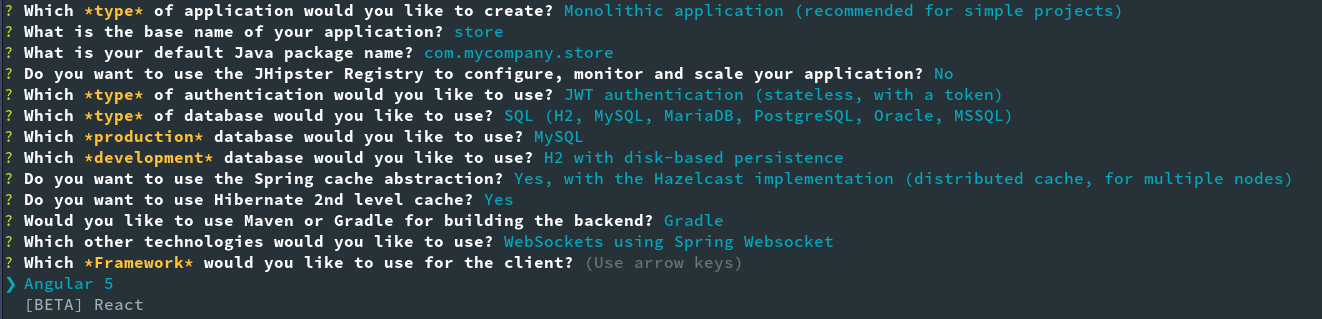
Domanda 10: questo prompt ci fornisce la scelta dello strumento di compilazione da utilizzare per il progetto; le opzioni sono Maven e Gradle. Scegliamo Gradle qui perché è più moderno e potente: (*NOI MAVEN*).



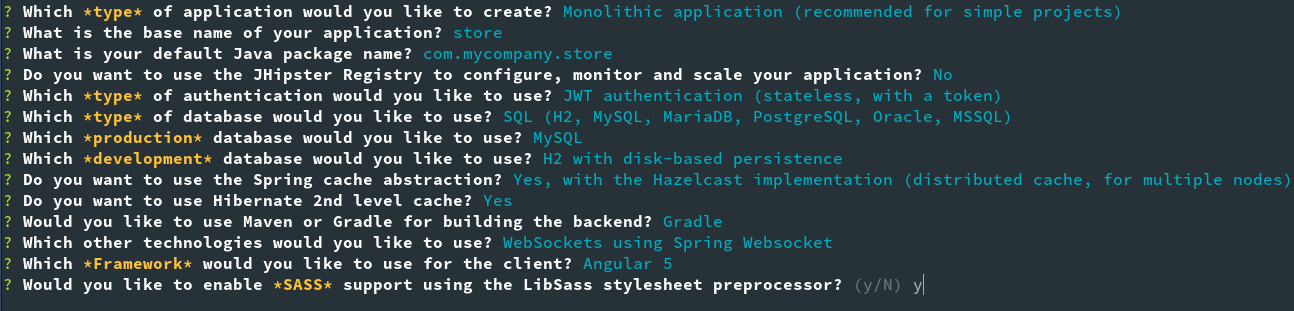
• Domanda 11: questo suggerimento è interessante in quanto presenta varie opzioni aggiuntive supportate da JHipster. Le opzioni sono:  
o Accesso social: aggiunge il supporto per l'utilizzo di un provider di accesso social come Facebook, Twitter e così via per l'accesso (l'opzione di accesso social viene rimossa in JHipster 5 e devi scegliere l'autenticazione OAuth 2.0 / OIDC invece di utilizzare l'accesso social fornito da OIDC fornitore)  
o Elasticsearch: aggiunge il supporto Elasticsearch per le entità generate  
o WebSockets: aggiunge il supporto WebSocket utilizzando il protocollo Spring WebSocket, SocketJS e Stomp  
o Primo sviluppo API con swagger-codegen: aggiunge il supporto codegen di Swagger per lo sviluppo dell'API  
o Apache Kafka: aggiunge il supporto per la coda asincrona usando Kafka  
Rendiamolo semplice e scegli WebSockets usando Spring WebSocket:



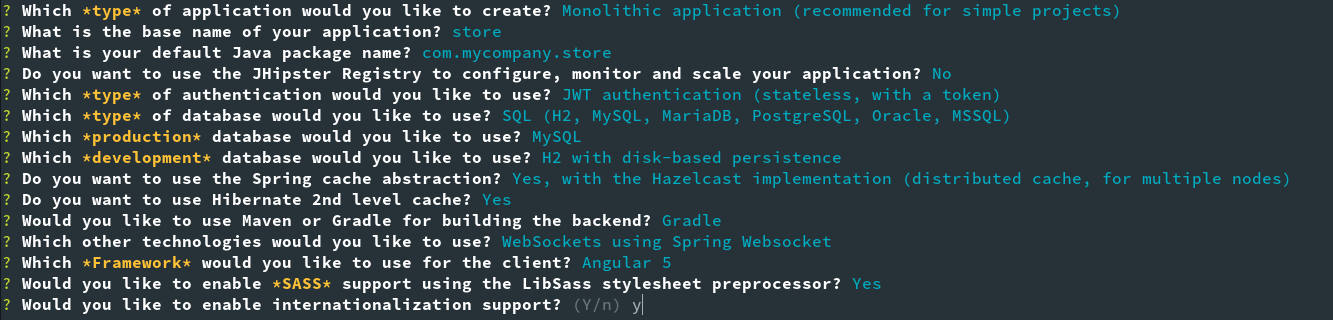
***Opzioni lato client***Ora, il generatore ci chiederà l'opzione lato client, incluso il framework lato client che desideriamo utilizzare:  
• Domanda 1: questo messaggio richiede di selezionare un framework MVVM lato client; le opzioni includono Angular 5 e React. Scegliamo Angular 5 qui:



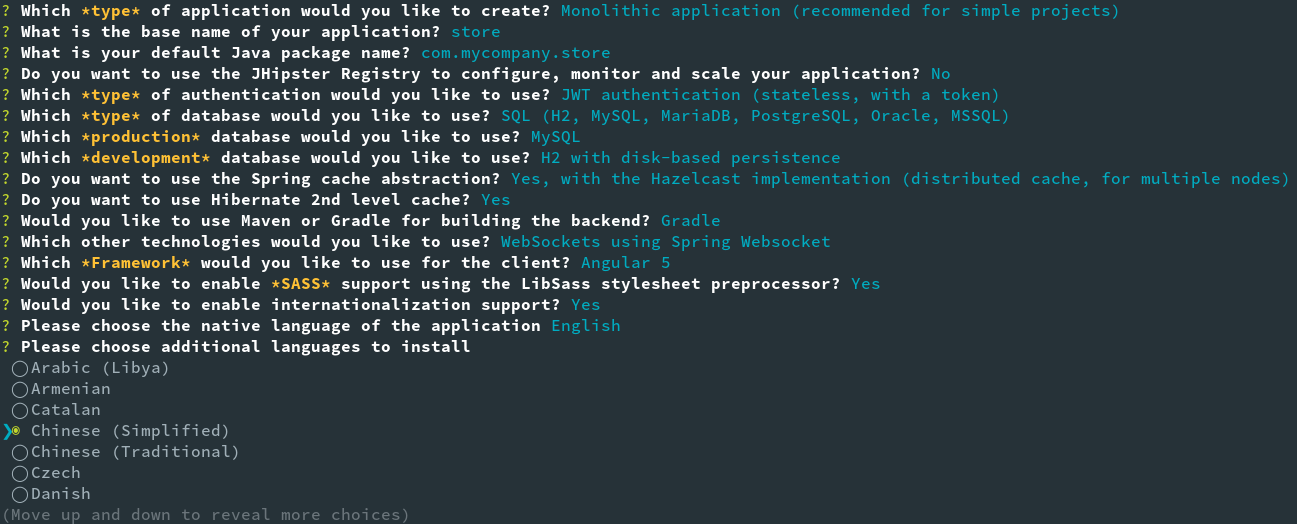
• Domanda 2. Questo prompt ci consente di abilitare il supporto SASS per il nostro CSS, e dal momento che SASS è fantastico, abilitarlo selezionando Sì:



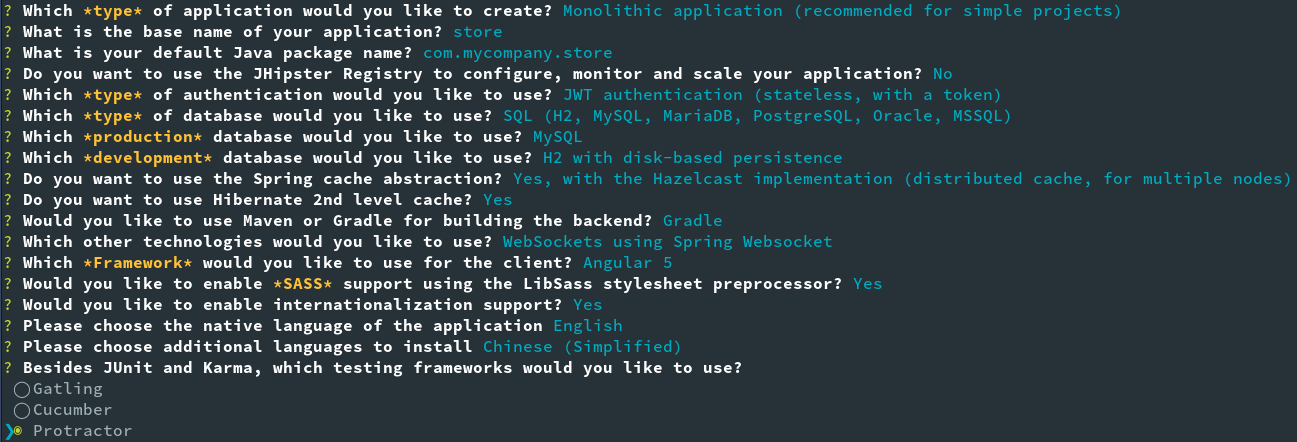
Opzioni di internazionalizzazione  
Ora avremo l'opportunità di abilitare l'internazionalizzazione e selezionare le lingue che vorremmo:  
• Domanda 1. Questo suggerimento ci consente di abilitare l'internazionalizzazione (i18n). Scegliamo Sì qui:



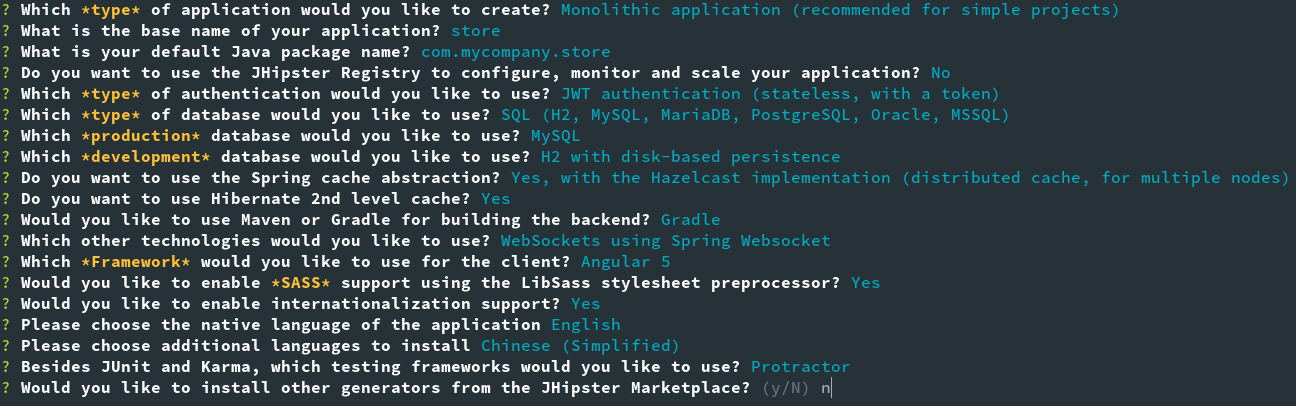
• Domanda 2: dal momento che abbiamo abilitato i18n, avremo la possibilità di scegliere una lingua principale e altre lingue i18n. Al momento della scrittura, ci sono 36 lingue supportate tra cui 2 lingue RTL (da destra a sinistra). Scegliamo l'inglese come lingua principale e il cinese semplificato come lingua aggiuntiva:



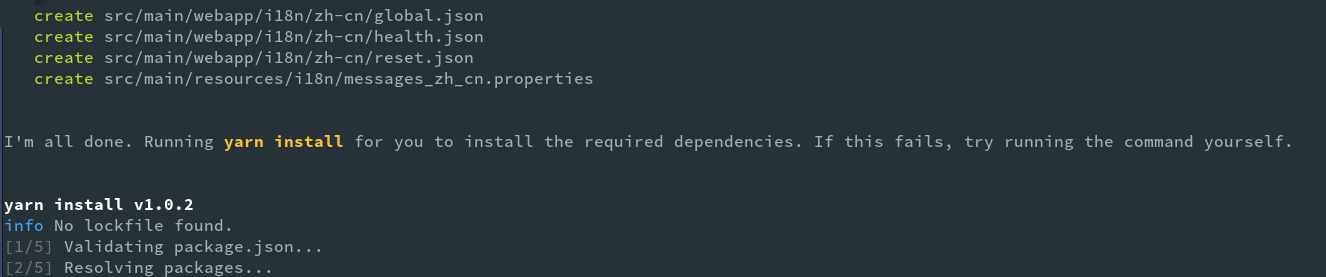
analisi  
Ora possiamo scegliere le opzioni di test per la nostra applicazione.  
Questo prompt ci consente di scegliere i framework di test per la nostra applicazione, che creeranno anche test di esempio per l'applicazione e le entità. Le opzioni sono Gatling, Cetriolo e Goniometro. Scegliamo Goniometro qui:



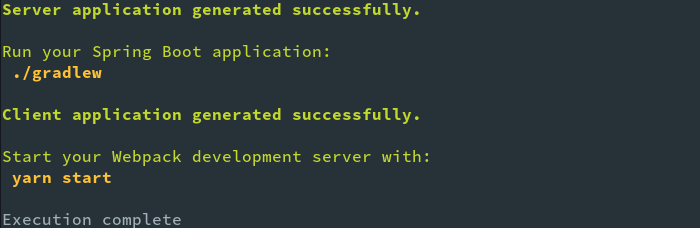
moduli  
Questo prompt ci consente di scegliere ulteriori moduli di terze parti dal marketplace JHipster (https://www.jhipster.tech/modules/marketplace). Questo può essere utile se vogliamo utilizzare funzionalità aggiuntive non supportate direttamente da JHipster. Lo vedremo nei prossimi capitoli. Per ora, scegliamo No. Non preoccuparti di questo, dato che questi moduli possono essere aggiunti all'applicazione in un secondo momento, quando richiesto:



Dopo aver risposto a tutte le domande, verrà avviata la generazione del codice e verrà visualizzato un output come segue, elencando i file creati e quindi eseguendo l'installazione del filo per ottenere l'installazione di tutte le dipendenze frontend.  
Se non si desidera che i passaggi di installazione di Yarn e Webpack vengano eseguiti, è possibile utilizzare il flag --skip-install durante l'esecuzione di JHipster per saltare questo. Basta eseguire jhipster --skip-install  
Una volta completata l'installazione, il generatore attiverà un build del webpack per il lato client in modo tale che quando avvieremo l'applicazione, avremo tutto pronto:



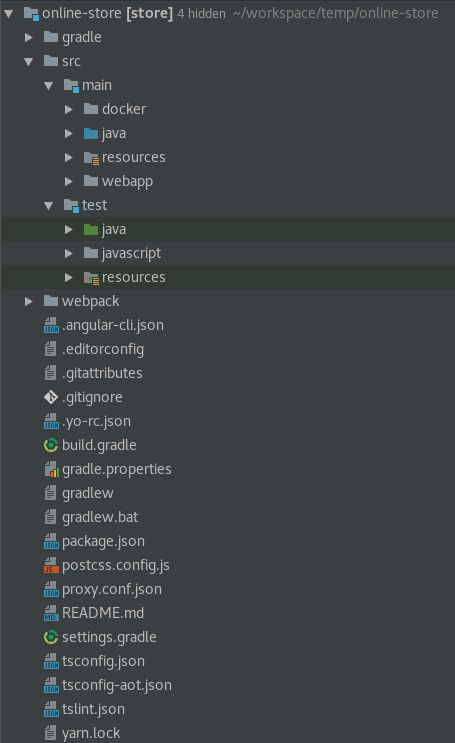
JHipster controllerà il tuo ambiente per vedere se sono installate tutte le dipendenze richieste come Java8, NodeJS, Git e NPM / Yarn. In caso contrario, mostrerà messaggi di avviso amichevoli prima che inizi la generazione del codice.  
Una volta completato il processo, verranno visualizzati i messaggi di successo come segue e le istruzioni per avviare l'applicazione:



Ci sono dei flag da riga di comando che possono essere passati durante l'esecuzione del comando jhipster. L'applicazione jhipster in esecuzione --help elencherà tutti i flag della riga di comando disponibili. Uno dei flag interessanti, ad esempio, è npm, che consente di utilizzare NPM anziché Yarn per la gestione delle dipendenze.  
JHipster inizializzerà automaticamente un repository Git per la cartella e impegnerà il file generato. Se desideri farlo tu stesso, puoi farlo passando il flag skip-git durante l'esecuzione di jhipster --skip-git ed esegui i passaggi manualmente come segue:  
> git init  
> git add --all  
> git commit -am "generato applicazione store online"  
Puoi anche utilizzare uno strumento GUI come Sourcetree o GitKraken se desideri farlo per lavorare con Git.

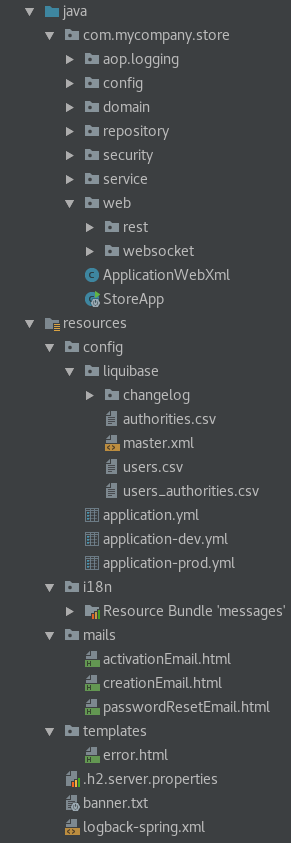
Code walkthrough

Ora che abbiamo generato la nostra applicazione con JHipster, passiamo attraverso parti importanti del codice sorgente che sono state create. Apriamo la nostra applicazione nel nostro IDE o Editor preferito.  
Se si utilizza IntelliJ IDEA, è possibile eseguire un'idea. in un terminale dalla cartella dell'applicazione per avviarlo. Altrimenti, puoi importare l'applicazione come un nuovo progetto Gradle usando l'opzione di menu File | Nuovo | Progetto da fonti esistenti e selezionare la cartella del progetto prima di selezionare Gradle dalle opzioni e fare clic su Avanti e quindi su Fine. Se stai usando Eclipse, apri File | Importa ... finestra di dialogo e seleziona Gradle Project nell'elenco e segui le istruzioni.  
Struttura del file  
L'applicazione creata avrà la seguente struttura di file:



Come puoi vedere, la cartella radice è piuttosto occupata con poche cartelle ma molti file di configurazione. Il più interessante tra loro è:  
• src: questa è la cartella di origine che contiene l'origine dell'applicazione principale e i file di origine del test.  
• webpack: questa cartella contiene tutte le configurazioni di build lato client Webpack per lo sviluppo, la produzione e il testing.  
• gradle: questa cartella include il wrapper Gradle e gli script di build Gradle aggiuntivi che verranno utilizzati dal file di build Gradle principale (JHipster fornisce un wrapper simile se si sceglie anche Maven).  
• build.gradle: questo è il nostro file di build Gradle che specifica il ciclo di vita delle nostre applicazioni. Ha anche le dipendenze lato server specificate. La build utilizza le proprietà definite nel file gradle.properties al suo fianco. È inoltre possibile trovare un eseguibile denominato gradlew (gradlew.bat per Windows), che consente di utilizzare Gradle senza installarlo.  
• .yo-rc.json: questo è il file di configurazione per JHipster. Questo file memorizza le opzioni selezionate durante la creazione di app e viene utilizzato per la rigenerazione e gli aggiornamenti delle app.  
• package.json: questo è il file di configurazione NPM che specifica tutte le dipendenze lato client, le dipendenze di compilazione lato client e le attività.  
• tsconfig.json: questa è la configurazione per Typescript. C'è anche la compilation tsconfig-aot.json per Angular AOT (Ahead-of-Time).  
• tslint.json: questa è la configurazione di lint per Typescript.  
Installa e configura Typescript e il plugin Tslint per l'IDE o l'editor per ottenere il massimo da Typescript.  
Ora, diamo un'occhiata alla cartella sorgente. Ha una cartella principale e una cartella di test, che contiene il codice sorgente dell'app principale e verifica il codice sorgente di conseguenza. La struttura della cartella è la seguente:  
• main:  
o docker: contiene il Dockerfile per l'applicazione e le configurazioni Docker per le opzioni selezionate  
o java: contiene il codice sorgente Java principale per l'applicazione  
o resources: contiene i file di configurazione Spring Boot, i changelog di Liquibase e le risorse statiche come i18n file lato server e i modelli di email utilizzati dall'applicazione  
o webapp: contiene il codice sorgente dell'applicazione Angular e il contenuto statico lato client come immagini, fogli di stile, file i18n e così via  
• Test:  
o java: contiene l'unità e la sorgente di test di integrazione per il lato server  
o javascript: contiene le specifiche del test dell'unità Karma e le specifiche end-to-end del goniometro per l'applicazione lato client  
o resources: contiene i file di configurazione Spring e le risorse statiche come i file i18n lato server e i modelli di email utilizzati dall'applicazione per il test

***Codice sorgente lato server***Il codice lato server si trova nella cartella Java e delle risorse sotto src / main, come mostrato nella schermata precedente. La struttura della cartella è la seguente:



È possibile notare che i componenti Spring non utilizzano le tradizionali annotazioni @Autowired o @Inject per l'integrazione delle dipendenze nel codice generato. Questo perché utilizziamo l'iniezione del costruttore invece dell'iniezione del campo e Spring Boot non ha bisogno di annotazioni esplicite per l'iniezione del costruttore. L'iniezione del costruttore è considerata migliore poiché consente di scrivere test unitari migliori ed evitare problemi di progettazione, mentre l'iniezione sul campo è più elegante ma rende facilmente monolitica una classe. L'iniezione del costruttore è una best practice suggerita dal team Spring. L'iniezione del costruttore rende inoltre più semplici i componenti dell'unità di test.  
Fonte Java  
Le parti importanti del codice sorgente Java sono:  
• StoreApp.java: questa è la classe principale per l'applicazione. Poiché si tratta di un'applicazione Spring Boot, la classe principale è eseguibile e puoi avviare l'applicazione eseguendo semplicemente questa classe da un IDE. Diamo un'occhiata a questa classe:  
o La classe è annotata con una serie di annotazioni Spring JavaConfig:

**@ComponentScan**  
**@EnableAutoConfiguration**(exclude = {MetricFilterAutoConfiguration.class, MetricRepositoryAutoConfiguration.class})  
**@EnableConfigurationProperties**({LiquibaseProperties.class, ApplicationProperties.class})

o Il primo, @ComponentScan, indica all'applicazione Spring di eseguire la scansione dei file di origine e di rilevare automaticamente i componenti Spring (Servizi, Repository, Risorse, Classi di configurazione che definiscono i bean Spring e così via).  
o Il secondo è @EnableAutoConfiguration, che indica a Spring Boot di provare a indovinare e autoconfigurare i bean di cui l'applicazione potrebbe aver bisogno in base alle classi trovate nel classpath e alle configurazioni fornite. Le impostazioni di esclusione specificano in particolare che Spring Boot non configura automaticamente i bean specificati.  
o Il terzo, @EnableConfigurationProperties, consente di registrare configurazioni aggiuntive per l'applicazione tramite file di proprietà.  
•  
o Il metodo principale della classe avvia l'applicazione Spring Boot e la esegue:

public static void main(String[] args) throws UnknownHostException {  
 SpringApplication app = new SpringApplication(StoreApp.class);  
 DefaultProfileUtil.addDefaultProfile(app);  
 Environment env = app.run(args).getEnvironment();  
 ...  
}

• config: questo pacchetto contiene configurazioni di bean Spring per database, cache, WebSocket e così via. Qui è dove configureremo varie opzioni per l'applicazione. Alcuni di quelli importanti sono:  
o CacheConfiguration.java: questa classe configura la cache di secondo livello di Hibernate per l'applicazione. Poiché abbiamo scelto Hazelcast come provider di cache, questa classe si configura allo stesso modo.  
o DatabaseConfiguration.java: questa classe configura il database per l'applicazione e abilita la gestione delle transazioni, il controllo JPA e i repository JPA per l'applicazione. Configura inoltre Liquibase per gestire le migrazioni di DB e il database H2 per lo sviluppo.  
*o SecurityConfiguration.java: questa è una parte molto importante dell'applicazione in quanto configura la sicurezza per l'applicazione. Diamo un'occhiata alle parti importanti della classe:  
♣ Le annotazioni consentono la sicurezza Web e la sicurezza a livello di metodo in modo che possiamo utilizzare le annotazioni @Secured e @ Pre / PostAuthorize su singoli metodi:*

@EnableWebSecurity  
@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)

*La seguente configurazione indica all'applicazione di ignorare il contenuto statico e alcune API dalla configurazione di sicurezza di Spring:*

@Override  
public void configure(WebSecurity web) throws Exception {  
 web.ignoring()  
 .antMatchers(HttpMethod.OPTIONS, "/\*\*")  
 .antMatchers("/app/\*\*/\*.{js,html}")  
 .antMatchers("/i18n/\*\*")  
 .antMatchers("/content/\*\*")  
 .antMatchers("/swagger-ui/index.html")  
 .antMatchers("/api/register")  
 .antMatchers("/api/activate")  
 .antMatchers("/api/account/reset-  
 password/init")  
 .antMatchers("/api/account/reset-  
 password/finish")  
 .antMatchers("/test/\*\*")  
 .antMatchers("/h2-console/\*\*");  
}

*La seguente configurazione indica a Spring Security quali endpoint sono consentiti per tutti gli utenti, quali endpoint devono essere autenticati e quali endpoint richiedono un ruolo specifico (ADMIN, in questo caso):*

@Override  
protected void configure(HttpSecurity http) throws Exception {  
 http  
 ...  
 .and()  
 .authorizeRequests()  
 .antMatchers("/api/register").permitAll()  
 ...  
 .antMatchers("/api/\*\*").authenticated()  
 .antMatchers("/websocket/tracker")  
 .hasAuthority(AuthoritiesConstants.ADMIN)  
 .antMatchers("/websocket/\*\*").permitAll()  
 .antMatchers("/management/health").permitAll()  
 .antMatchers("/management/\*\*")  
 .hasAuthority(AuthoritiesConstants.ADMIN)  
 .antMatchers("/v2/api-docs/\*\*").permitAll()  
 .antMatchers("/swagger-resources/configuration/ui").permitAll()  
 .antMatchers("/swagger-ui/index.html")  
 .hasAuthority(AuthoritiesConstants.ADMIN)  
 .and()  
 .apply(securityConfigurerAdapter());  
}

o WebConfigurer.java: qui è possibile configurare le intestazioni della cache HTTP, i mapping MIME, la posizione delle risorse statiche e CORS (Cross-Origin Resource Sharing).

JHipster offre un ottimo supporto CORS fuori dalla scatola:  
• CORS può essere configurato utilizzando la proprietà jhipster.cors, come definito nelle proprietà comuni dell'applicazione JHipster (http://www.jhipster.tech/common-application-properties/).  
• È abilitato di default in modalità dev per monoliti e gateway. È disabilitato per impostazione predefinita per i microservizi in quanto si suppone che accedano tramite un gateway.  
• Per impostazione predefinita, è disabilitato in modalità prod per monoliti e microservizi, per motivi di sicurezza.  
• domain: le classi del modello di dominio per l'applicazione si trovano in questo pacchetto. Questi sono POJO semplici che hanno annotazioni JPA che lo associano a un'entità di Hibernate. Quando l'opzione Elasticsearch è selezionata, anche questi fungono da oggetto Document. Diamo un'occhiata alla classe User.java:  
o Una classe di entità è caratterizzata dalle seguenti annotazioni. L'annotazione @Entity contrassegna la classe come entità JPA. L'annotazione @Table mappa l'entità in una tabella di database. L'annotazione @Cache abilita il caching di secondo livello dell'entità e specifica anche una strategia di caching:

@Entity

@Table(name = "jhi\_user")

@Cache(usage = CacheConcurrencyStrategy.NONSTRICT\_READ\_WRITE)

• Ci sono varie annotazioni usate a livello di campo in queste classi. @Id indica la chiave primaria per l'entità. @Column esegue il mapping di un campo a una colonna della tabella del database con lo stesso nome quando non viene fornita alcuna sovrascrittura. @NotNull, @Pattern e @Size sono annotazioni utilizzate per la convalida. @JsonIgnore viene utilizzato da Jackson per ignorare i campi durante la conversione degli oggetti in JSON che devono essere restituiti nelle richieste dell'API REST. Ciò è particolarmente utile con Hibernate in quanto evita riferimenti circolari tra le relazioni, che creano tonnellate di richieste del DB SQL e falliscono:

@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;  
  
@NotNull  
@Pattern(regexp = Constants.LOGIN\_REGEX)  
@Size(min = 1, max = 50)  
@Column(length = 50, unique = true, nullable = false)  
private String login;  
  
@JsonIgnore  
@NotNull  
@Size(min = 60, max = 60)  
@Column(name = "password\_hash",length = 60)  
private String password;

o ***Le relazioni tra le tabelle del database sono anche mappate alle entità usando annotazioni JPA. Qui, ad esempio, mappa una relazione molti-a-molti tra un utente e le autorità dell'utente. Specifica inoltre una tabella di join da utilizzare per la mappatura:***

@JsonIgnore  
@ManyToMany  
@JoinTable(  
 name = "jhi\_user\_authority",  
 joinColumns = {@JoinColumn(name = "user\_id", referencedColumnName = "id")},  
 inverseJoinColumns = {@JoinColumn(name = "authority\_name", referencedColumnName = "name")})  
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT\_READ\_WRITE)  
@BatchSize(size = 20)  
private Set<Authority> authorities = new HashSet<>();

• repository: questo pacchetto contiene i repository Spring Data per le entità. Queste definizioni di interfaccia tipicamente vengono implementate automaticamente da Spring Data. Ciò elimina la necessità per noi di scrivere qualsiasi implementazione dello standard per il livello di accesso ai dati. Diamo un'occhiata all'esempio UserRepository.java:

@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
  
 Optional<User> findOneByActivationKey(String activationKey);  
  
 List<User> findAllByActivatedIsFalseAndCreatedDateBefore(Instant   
 dateTime);  
  
 Optional<User> findOneByResetKey(String resetKey);  
  
 Optional<User> findOneByEmailIgnoreCase(String email);  
 ...  
}

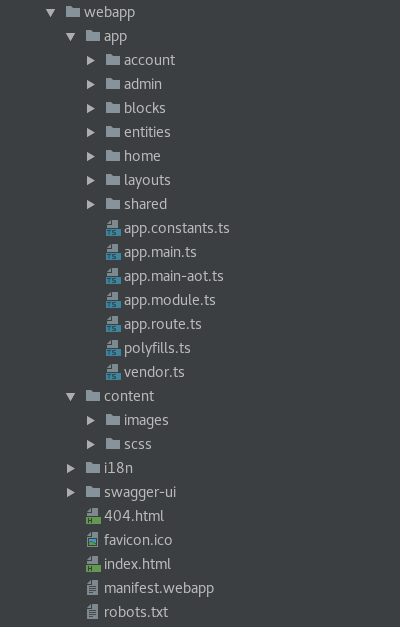
***o L'annotazione @Repository contrassegna ciò come un componente del repository dati Spring.  
o L'interfaccia estende JpaRepository, che consente di ereditare tutte le operazioni CRUD predefinite come findOne, findAll, save, count ed delete.  
o I metodi personalizzati vengono scritti come definizioni di metodi semplici seguendo le convenzioni di denominazione dei dati Spring in modo che il nome del metodo specifichi la query da generare. Ad esempio, findOneByEmailIgnoreCase genera una query equivalente a SELECT \* FROM user WHERE LOWER (email) = LOWER (: email).  
• security: questo pacchetto contiene componenti e utilità relativi alla sicurezza di Spring e, poiché abbiamo scelto JWT come meccanismo di autenticazione, contiene anche classi correlate a JWT come TokenProvider, JWTFilter e JWTConfigurer.  
• service: questo pacchetto contiene il livello di servizio costituito da bean di servizi Spring, DTO, mapper di Mapstruct DTO e utility di servizio.  
• web: questo pacchetto contiene le classi di risorse Web, visualizza classi di modelli e classi di utilità.  
o rest: questo pacchetto contiene classi di risorse Spring per l'API REST. Inoltre, contiene oggetti e utilità del modello di visualizzazione. Diamo un'occhiata a UserResource.java:  
♣ Le classi resource sono contrassegnate con le annotazioni @RestController e @RequestMapping ("/ api") di Spring. Quest'ultima specifica il percorso dell'URL di base per il controller in modo che tutte le richieste <applicationContext> / api / \* siano inoltrate a questa classe.  
♣ I metodi di richiesta sono annotati con annotazioni in base al loro scopo, per esempio, il sotto segna il metodo createUser come PostMapping per "/ users", il che significa che tutte le richieste POST a <applicationContext> / api / users saranno servite da questo metodo. L'annotazione @Timed viene utilizzata per misurare*** le prestazioni del metodo. L'annotazione @Secured limita l'accesso al metodo al ruolo specificato:

**@PostMapping("/users")**  
**@Timed**  
**@Secured**(AuthoritiesConstants.ADMIN)  
public ResponseEntity createUser(@Valid @RequestBody ManagedUserVM managedUserVM) throws URISyntaxException {  
 ...  
}

o WebSocket: questo pacchetto contiene i controller Websocket e visualizza i modelli.  
*JHipster utilizza DTO (Data Transfer Object) e VM (View Model) sul lato server. Le DTO servono per trasferire i dati dal livello di servizio a e dal livello risorse. Interrompono le transazioni di Hibernate ed evitano ulteriori carichi pigri dall'essere attivati dal livello di risorse. Le macchine virtuali vengono utilizzate solo per la visualizzazione dei dati sul front-end Web e non interagiscono con il livello di servizio.*

**Resources**

Le parti importanti delle risorse sono:  
• config: contiene i file YAML e il changelog di Liquibase. Il file application.yml contiene Spring Boot, JHipster e le proprietà specifiche dell'applicazione durante l'applicazione. (Dev | prod) I file .yml contengono proprietà che devono essere applicate quando il profilo dev o prod specifico è attivo. Le configurazioni di test sono in src / test / resource / application.yml.  
• i18n: contiene i file di risorse i18n sul lato server.  
• mails: contiene i modelli di Thymeleaf per le email.  
• templates: contiene modelli di Thymeleaf per il lato client.  
codice sorgente lato client  
Il codice sorgente lato client si trova nella cartella src / main / webapp, come abbiamo visto in precedenza. La struttura è la seguente:



I più degni di nota tra questi sono:  
• app: questa cartella contiene il codice sorgente Typescript dell'applicazione Angular, che è organizzato con una cartella per caratteristica:  
o app.main.ts: questo è il file principale per l'app Angular. Questo avvia l'applicazione Angolare. Si noti che utilizza platformBrowserDynamic, che consente all'applicazione di funzionare con la compilazione JIT (Just-in-time) nel browser. Questo è l'ideale per lo sviluppo:

platformBrowserDynamic().bootstrapModule(StoreAppModule)  
.then((success) => console.log(`Application started`))  
.catch((err) => console.error(err));

• app.module.ts: questo è il modulo principale per l'app Angular. Dichiara componenti e fornitori a livello di app e importa altri moduli per l'applicazione. Avvia inoltre il componente principale dell'applicazione:

@NgModule({  
 imports: [  
 BrowserModule,  
 ...  
 StoreEntityModule,  
 // jhipster-needle-angular-add-module JHipster   
 will add new module here  
 ],  
 declarations: [  
 JhiMainComponent,  
 ...  
 FooterComponent  
 ],  
 providers: [  
 ProfileService,  
 ...  
 UserRouteAccessService  
 ],  
 bootstrap: [ JhiMainComponent ]  
})  
export class StoreAppModule {}

-account: questo modulo è costituito da funzionalità relative all'account quali attivazione, password, reimpostazione della password, registro e impostazioni. Ogni componente tipico è costituito dai file component.html, component.ts, route.ts e service.ts.  
• admin: questo modulo è costituito da funzionalità relative all'amministrazione quali audit, configurazione, documentazione, stato, registri, metriche, tracker e gestione degli utenti. Ogni componente tipico è costituito dai file component.html, component.ts, route.ts e service.ts.  
o blocks: questa cartella è costituita da intercettori HTTP e altre configurazioni utilizzate dall'applicazione.  
o entities: è qui che verranno creati i moduli di entità.  
o home: il modulo homepage.  
o layouts: questa cartella ha componenti di layout come la barra di navigazione, il piè di pagina, le pagine di errore e così via.  
o shared: questo modulo contiene tutti i servizi condivisi (auth, tracker, user), componenti (login, alert), modelli di entità e utilità richiesti per l'applicazione.  
• content: questa cartella contiene contenuti statici come immagini, CSS e file SASS.  
• i18n: è qui che vivono i file JSON i18n. Ogni lingua ha una cartella con numerosi file JSON organizzati da moduli.  
• swagger-ui: questa cartella ha il client UI Swagger utilizzato in fase di sviluppo per la documentazione API.  
• index.html: questo è il file di indice dell'applicazione Web. Questo contiene un codice molto minimale per caricare il componente principale dell'applicazione angolare. Si tratta di un'applicazione angolare a pagina singola. Troverai anche qualche codice di utilità commentato come lo script di analisi di Google e gli script di lavoro del servizio su questo file. Questi possono essere abilitati se richiesto:

<!doctype html>  
<html class="no-js" lang="en" dir="ltr">  
<head>  
 ...  
</head>  
<body>  
 ...  
 **<jhi-main></jhi-main>**  
 <noscript>  
 <h1>You must enable javascript to view this page.</h1>  
 </noscript>  
 ...  
</body>  
</html>

Per abilitare la modalità PWA usando i service workers, è sufficiente decommentare il codice corrispondente in src / main / webapp / index.html per registrare il service workers. JHipster utilizza la casella di lavoro (https://developers.google.com/web/tools/workbox/), che crea il rispettivo lavoratore del servizio e genera dinamicamente sw.js.

***Avvio dell'applicazione***

Ora, iniziamo l'applicazione e vediamo l'output. Esistono diversi modi per eseguire l'applicazione:

* Usando Spring Boot Gradle task dal terminale / riga di comando
* Eseguendo la classe principale Java src/main/java/com/mycompany/store/StoreApp.java da un IDE
* Eseguendo il file dell'applicazione pacchettizzato usando il comando java -jar

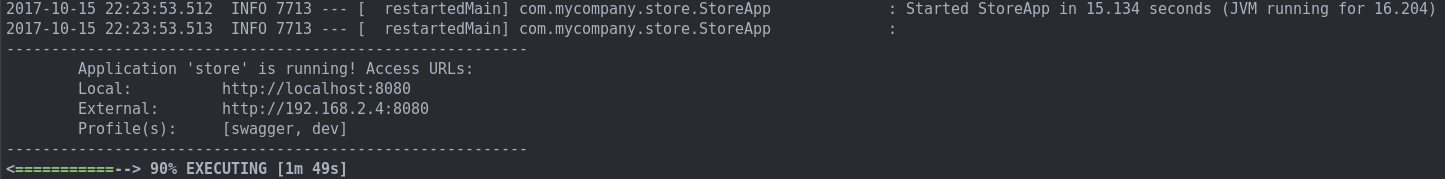
niziamo l'applicazione usando Gradle task. Se si desidera eseguire l'applicazione direttamente nell'IDE, è sufficiente aprire il file dell'app principale StoreApp.java menzionato in precedenza, fare clic con il pulsante destro del mouse e scegliere Run "StoreApp".

Per avviare l'applicazione tramite Gradle, apri un terminale / riga di comando e vai alla cartella dell'applicazione. Quindi, esegui il comando Gradle come segue (se sei su Windows, esegui gradlew.bat). Questo attiverà l'attività di default bootRun:

**> cd online-store  
> ./gradlew**

Esecuzione ./gradlew equivale all'esecuzione ./gradlew bootRun -Pdev. Per il lato client, la build del webpack deve essere eseguita prima di avviare il server per la prima volta, altrimenti vedrete una pagina vuota. Questa attività viene eseguita automaticamente durante la generazione dell'app, ma se per qualche motivo fallisce, può essere attivata manualmente eseguendo il webpack run run: build. L'attività può essere attivata direttamente anche dal comando Gradle eseguendo ./gradlew webpackBuildDev bootRun -Pdev.

Gradle inizierà a scaricare il wrapper e le dipendenze e dovresti vedere l'output della console simile allo screenshot seguente dopo un po 'di tempo (da qualche secondo a qualche minuto a seconda della velocità della rete):



L'app è stata avviata correttamente ed è disponibile su http: // localhost: 8080. Apri il tuo browser preferito e naviga fino all'URL.

Si noti che la generazione precedente rimarrà al 90% mentre il processo è in esecuzione in modo continuo

***Moduli di applicazione***

Vediamo i diversi moduli disponibili fuori dalla scatola. I moduli possono essere raggruppati in:

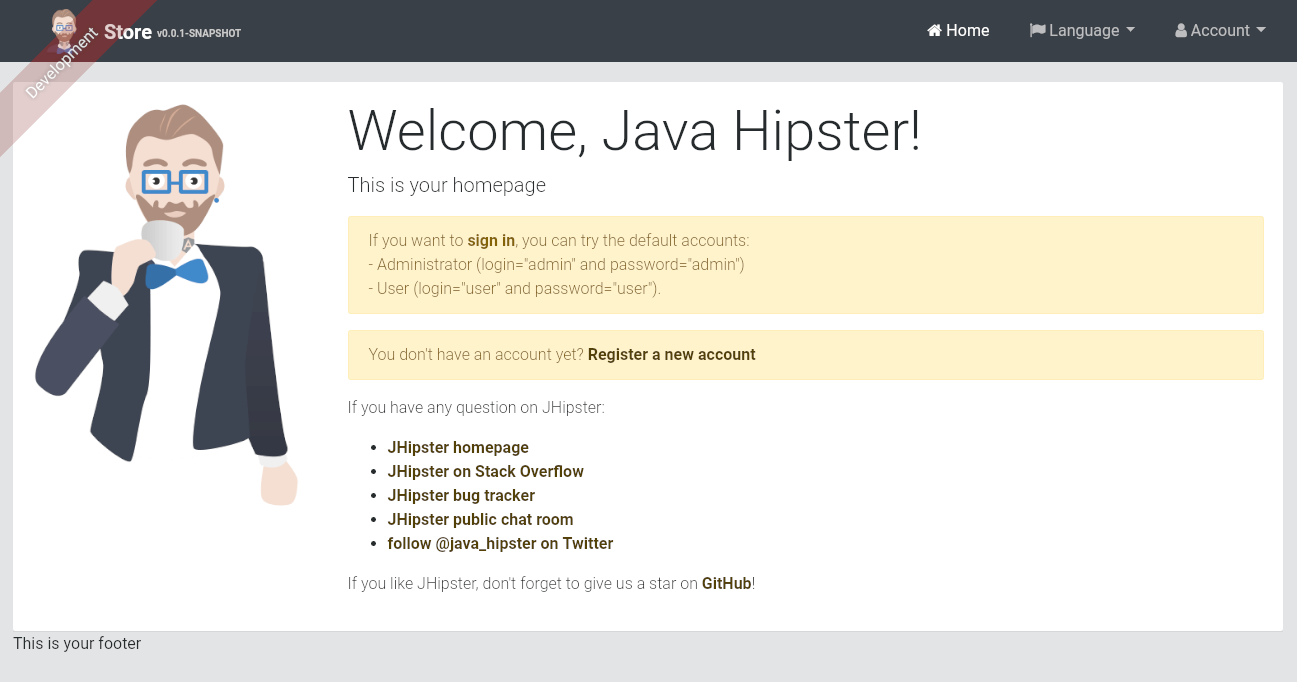
• Home and Login

• Account

• Admin

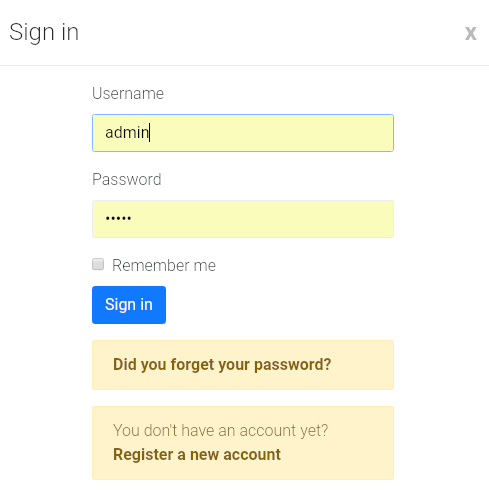
***Moduli Home e Login***

Una volta aperto l'URL, vedrai un hipster alla moda bere caffè sulla homepage come segue:

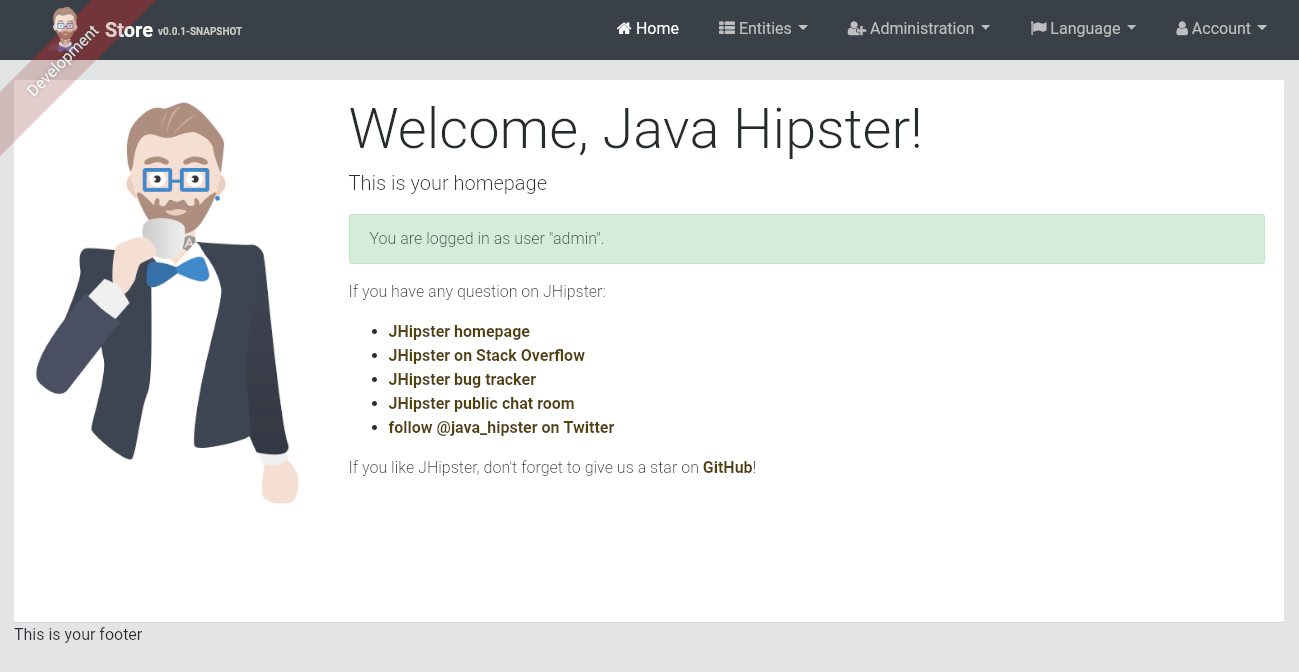


Questa è la home page. Accedere all'applicazione utilizzando le credenziali predefinite.

1. Fare clic sul collegamento Accedi nella pagina o Account | Accedi. Verrà visualizzata la seguente schermata di accesso. Inserisci le credenziali predefinite-Username-admin, Password-admin e fai clic su Accedi:



Una volta effettuato l'accesso, vedrai la home page autenticata con tutte le voci di menu autenticate nella barra di navigazione:



1. Dal momento che abbiamo abilitato l'internazionalizzazione, otteniamo un menu Lingua. Proviamo a passare a una lingua diversa. Fai clic sul menu Lingua e scegli la prossima lingua disponibile:

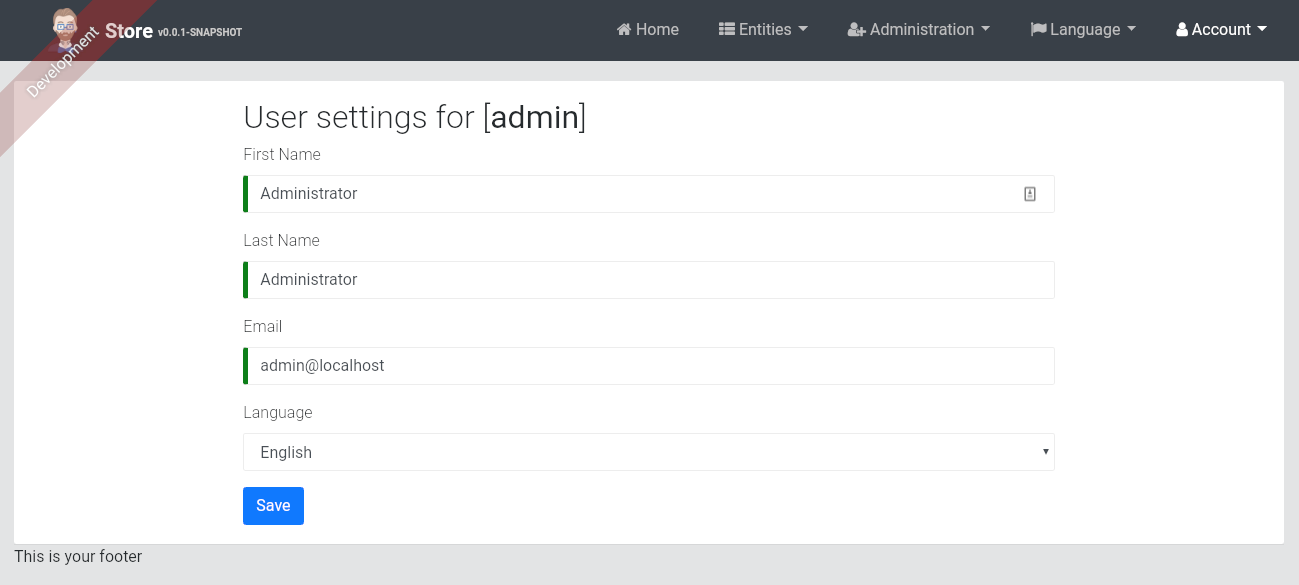
***Moduli account***

Ora, diamo un'occhiata ai moduli account creati fuori dagli schemi. Nel menu Account, vedrai un'opzione di disconnessione e i seguenti moduli:

* Settings
* Password
* Registration

# Settings

Questo modulo ti consente di modificare le impostazioni dell'utente come nome, email e lingua:



*Password*

Continuare da pag. 64