Entwurfsbeschreibung

Version	Autoren	Datum	Kommentar
1.0	NH, RR, PF, AW	2006-05-22	

Inhaltsverzeichnis

1 Allgemeines

Die Anwendung teamInstance ist ein Werkzeug, welches den arbeitsteiligen Zugriff auf OWL-basierte Ontologien und darauf aufbauender Wissensbasen unterstützt. Hierzu nutzt teamInstance eine Client-Server-Architektur, in der die Clients über eine Webbrowser-Schnittstelle auf die serverseitige zentrale Datenhaltung zugreifen.

2 Produktübersicht

Die Applikation wird als JSR-168-kompatible Portlet-Applikation entwickelt und als solche in ein Jetspeed-2-Portal integriert. Es enthält

- ein Rechtesystem für den arbeitsteiligen Umgang mit Wissensbasen,
- Portlets zur Darstellung der Applikationslogik auf Clientebene sowie
- ein objektorientiertes und erweiterbares Applikationsmodell.

3 Grundsätzliche Struktur- und Entwurfsprinzipien des Gesamtsystems

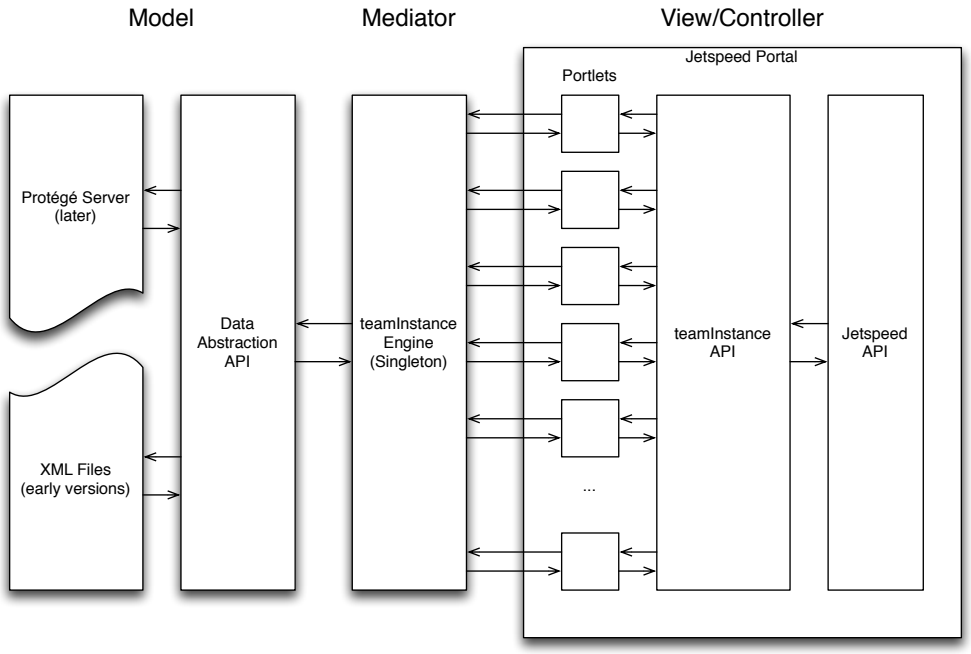

Abbildung 1: teamInstance-Architektur nach dem Model-2-Prinzip

3.1 teamInstance-Architektur

Leitgedanke beim Entwurf der teamInstance-Architektur war das OO-Prinzip *offen für Erweiterungen aber verschlossen gegenüber Modifikationen.* Um die Erweiterbarkeit des Gesamtsystems ohne Modifikation der Kernkomponenten zu gewährleisten wurde deshalb eine 3-Schichten-Architektur entworfen, die sich beiderseits in die vorgegebenen Architekturen von Jetspeed und der Jena-API bzw. InstanceXLs einpasst. Um bezüglich der Kernkomponenten (Portlets, Engine) unabhängig von konkreten Implementierungen zu bleiben, wurde ein weiteres wichtiges OO-Prinzip umgesetzt: *Programmierung gegen Schnittstellen und nicht gegen Implementierungen.* So wurden 2 Abstraktionsschichten entworfen, die zwischen Jetspeed-Portal und Datenhaltung geschoben eine Programmierung gegen Schnittstellen ermöglichen und somit die Implementierung der Kernkomponenten von den konkreten Implementierungen Jetspeeds bzw. der Datenhaltung unabhängig machen. Einen Überblick über die Architektur des Gesamtsystems gibt Abbildung 1 (Seite 2).

3.2 Aufteilung der Systemkomponenenten in Pakete

Die Aufteilung der Systemkomponenten auf verschiedene Pakete wurden wie folgt vorgenommen:

Datenabstraktion de.teaminstance.om.datamodel

Datenmodell, Engine de.teaminstance.om

View- und Controller-Komponenten de.teaminstance.portlets

Abstraktionschicht zur Jetspeed-API de.teaminstance.security

Weiterhin sind noch Pakete für Exceptions (de.teaminstance.exceptions) und verschiedenen, z. T statische Hilfsklassen (de.teaminstance.util) erstellt worden. Abbildung 2 zeigt das Paketdiagramm der teamInstance-Applikation.

3.3 Benutzungsoberfläche

Die Benutzungsoberfläche ist zum Teil durch Jetspeed-2 und die Browser-Schnittstelle vorgegeben, wird aber so weit wie möglich durch Cascading Style Sheets (CSS) und evtl. JavaScript an die Benutzungsoberfläche von InstanceXL bzw. Protégé angepasst. Abbildung 3 stellt noch nicht das endgültige Erscheinungsbild von teamInstance dar, soll jedoch bereits einen groben Überblick darüber geben, wie teamInstance später mit dem Benutzer interagiert. Dafür gewählt wurde die Ansicht, welche sich dem Benutzer beim Öffnen eines Projektes präsentiert, da der Umgang mit Projekten die Kernfunktionalität von teamInstance darstellt. Login- bzw. Logout wird absichtlich nicht durch ein Portlet realisiert, sondern ist ähnlich dem elatePortal am oberen Fensterrand untergebracht, um die Übersichtlichkeit zu erhöhen und dem Benutzer zu jeder Zeit direkt zur Verfügung zu stehen.

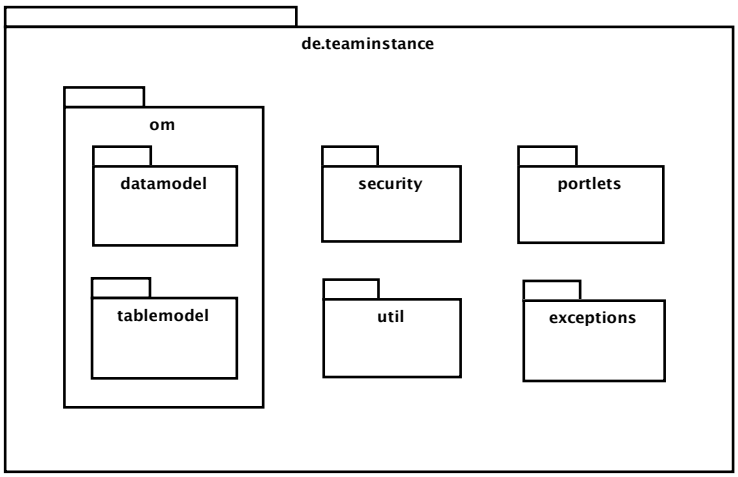

Abbildung 2: Paketdiagramm

3.4 Typen von PSML-Seiten

Es wird in teamInstance für die verschiedenen Situationen unterschiedliche Typen von PSML-Seiten geben, von denen die wichtigsten im Folgenden aufgeführt und erläutert werden.

Startseite, unangemeldet
Diese Seite bekommt ein Benutzer zu sehen, wenn er teamInstance nutzen möchte, aber nicht am System angemeldet ist. Sie besteht aus

- einem Login-Formular am oberen Fensterrand sowie
- einem Portlet, welches den Benutzer zum Anmelden auffordert.

Startseite, angemeldet
Diese Seite bekommt ein Benutzer zu sehen, wenn er sich gerade am System angemeldet hat. Sie besteht aus

- einem Abschnitt am oberen Fensterrand, der den Benutzernamen sowie einen Logout-Link und einen Link zu einer Accountseite anzeigt,
- einem Projektbrowser-Portlet, welches sämtliche für den Benutzer verfügbaren Projekte mit ID, Namen und Rechten auflistet und des Weiteren die Möglichkeit bietet, Projekte aus dieser Liste zu öffnen sowie
- einem Portlet, welches Informationen zum Benutzer anzeigt, z. B. den Zeitpunkt der letzten Anmeldung.

Projektseite
Diese Seite wird dynamisch erzeugt und repräsentiert ein geöffnetes Projekt. Sie wird dem Benutzer als zusätzlicher Tab angezeigt und besteht aus

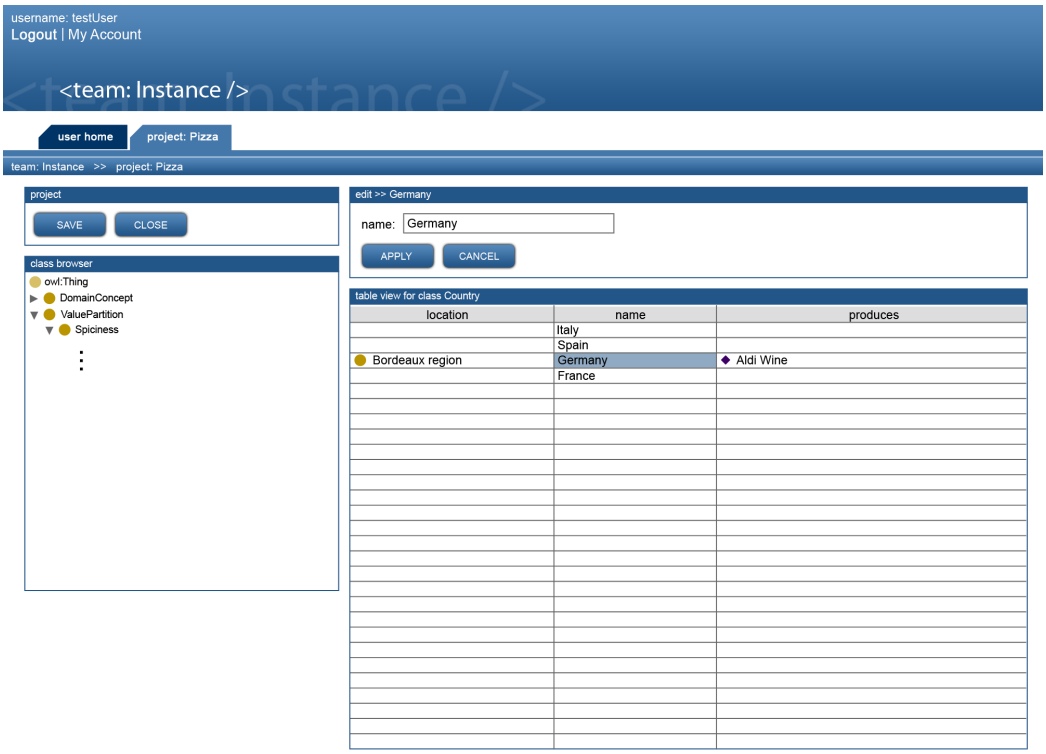

Abbildung 3: prototypische Benutzungsoberfläche von teamInstance

- einem Portlet mit Funktionen zum Speichern und Schließen eines Projektes,
- einem Klassenbrowserportlet zum Anzeigen aller benutzerdefinierten Klassen im geöffneten Projekt,
- einem Tabellenportlet zum Anzeigen aller Instanzen einer im Klassenbrowser ausgwählten Klasse sowie
- einem Editorportlet zum Ändern des Wertes in einer zuvor ausgewählten Tabellenzelle.

Accountseite
Diese Seite wird temporär angezeigt, wenn der Benutzer den entsprechenden Link am oberen Fensterrand auswählt. Sie präsentiert sich als zusätzlicher Tab und besteht aus einem Portlet zum Ändern des Passwortes und der gespeicherten E-Mail-Adresse.

Administrationsseite, Benutzerverwaltung
Diese Seite wird lediglich einem Benutzer mit der Rolle Adminstrator angezeigt. Sie besteht aus

- einem Portlet zum Registrieren von neuen Benutzern,
- einem Portlet, welches eine Liste aller registrierten Benutzer anzeigt sowie

- einem Portlet zum Ändern von Benutzerattributen bzw. zum Löschen von Benutzern.

Administrationsseite, Projektverwaltung
Diese Seite wird lediglich einem Benutzer mit der Rolle Adminstrator angezeigt. Sie besteht aus

- einem Portlet zum Erstellen eines neuen Projektes aus einer bestehenden Wissensbasis sowie
- einem Portlet zum Bearbeiten eines Projektes, einschl. der Verwaltung der Benutzer zu einem Projekt.

In Abbildung 4 ist die Logik der Seitenaufrufe am Portal ersichtlich.

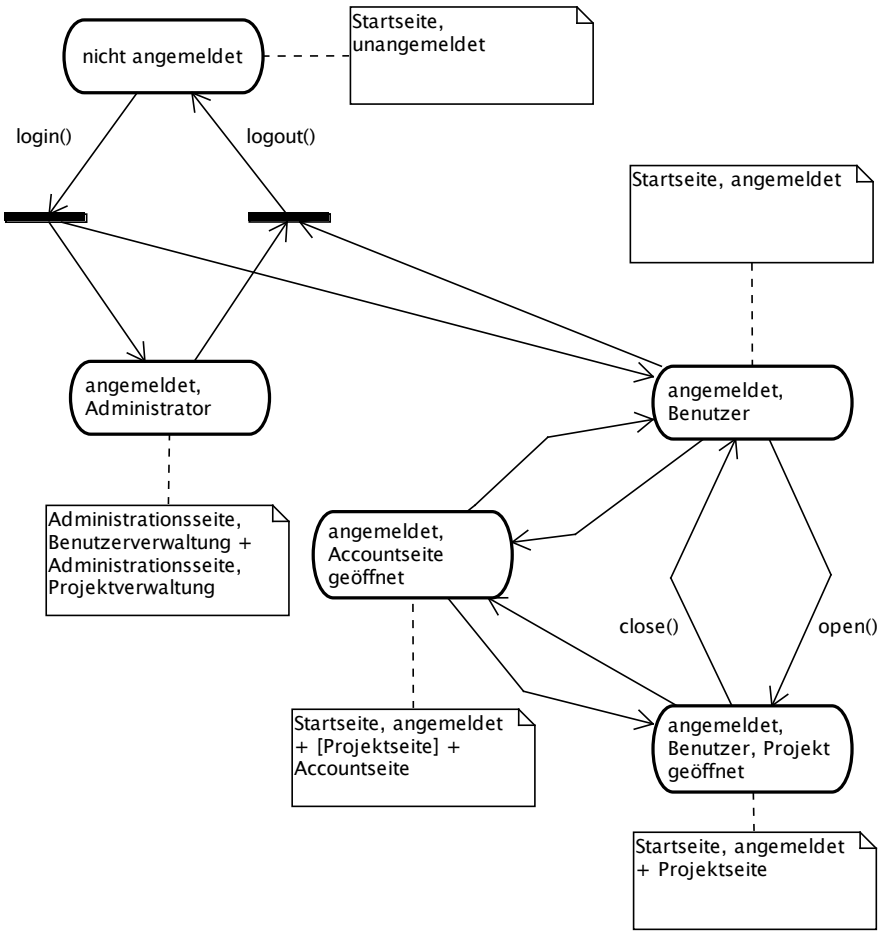

Abbildung 4: Seitenaufrufe am Portal

3.5 Entwurfsmuster

3.5.1 Model 2

Das für Webanwendungen angepasste MVC-Metamuster bezeichnet man als Model 2. Hierbei ist die strenge Aufteilung in 3 separeate Klassen etwas aufgeweicht, da View und Controller zusammen ein Portlet bilden, wobei die Verantwortlickeiten aber auf die Portletklasse als Controller und die darstellende JSP als View aufgeteilt sind. Außerdem existiert eine Vermittlerklasse, die gewöhnlich als JavaBean implementiert wird. Über sie können die einzelnen Controller auf die gewünschten Daten aus dem Model zugreifen.

3.5.2 Singleton

Das Singleton-Muster findet Anwendung wenn sicher gestellt werden soll, dass zu jedem Zeitpunkt höchstens eine Instanz einer bestimmten Klasse existiert. Dazu enthält die ensprechende Klasse eine Klassenvariable ihres eigenen Typs, die sie auf Anfrage zurückliefert und einen privaten Konstruktor, der ein externes Instanziieren unmöglich macht.

3.5.3 Adapter

Um eine komplexe Schnittstelle in eine einfachere zu transformieren, verwendet man das Adapter-Muster. Hierbei wird eine Klasse zwischengeschaltet, die über ein einfaches Spektrum öffentlicher Methoden verfügt, welche einfach zu benutzen sind. Diese Klasse prozessiert die Anfragen, indem sie intern mehrere Methoden aufruft, die sogar auf verschiedene Objekte verteilt sein können. Das Adapter-Muster führt zu einer einfacheren Implementierbarkeit und bei sinnvoller Anwendung und Benennung zu leichter lesbarem Code.

3.5.4 Mediator

Komplexe Kommunikation zwischen Klassen lässt sich am besten mittels des Mediator-Musters modellieren. Dabei agiert eine Klasse als Vermittler zwischen beliebig vielen anderen Klassen. Durch dieses Muster erreicht man eine lose Kopplung der Klassen, die dadurch änderungsfreundlicher und besser wartbar werden.

4 Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

4.1 de.teaminstance.om

Das Paket de.teaminstance.om ist das zentrale Paket der Anwendung. Es enthält Klassen zur Modellierung abstrakter Objekte wie `TIProject` oder `TIProjectManager`. Außerdem enthält es zwei Unterpakete: de.teaminstance.datamodel und de.teaminstance.tablemodel. Im Paket datamodel befinden sich Klassen wie `TIKnowledgeBase`, die vor allem die zu speichernden Daten implementierungsunabhängig modellieren.

Das om-Paket verwendet das Singleton-Muster (siehe Abschnitt 3.5.2), um sicherzustellen, dass von der Klasse TIEngine maximal eine Instanz existiert. Außerdem fungiert TIEngine als Vermittler zwischen den Controller-Portlets und dem Datenmodell (vgl. Abschnitt 3.5.4).

4.2 de.teaminstance.exceptions

Dieses Paket enthält alle Exceptions, die nicht von der J2SE durch Klassen aus den Paketen `java.lang.Exception` oder `java.io` zur Verfügung gestellt, aber von teamInstance benötigt werden. Die Klasse `TIException` beinhaltet sämtliche Funktionalität, welche allen teamInstance-Exceptions gemein ist, und wird von `java.lang.Exception` abgeleitet. Sämtliche teamInstance-spezifischen Exceptions, bspw. `ProjectNotFoundException` oder `UnrecognizedFileFormatException`, werden dann von `TIException` abgeleitet.

4.3 de.teaminstance.portlets

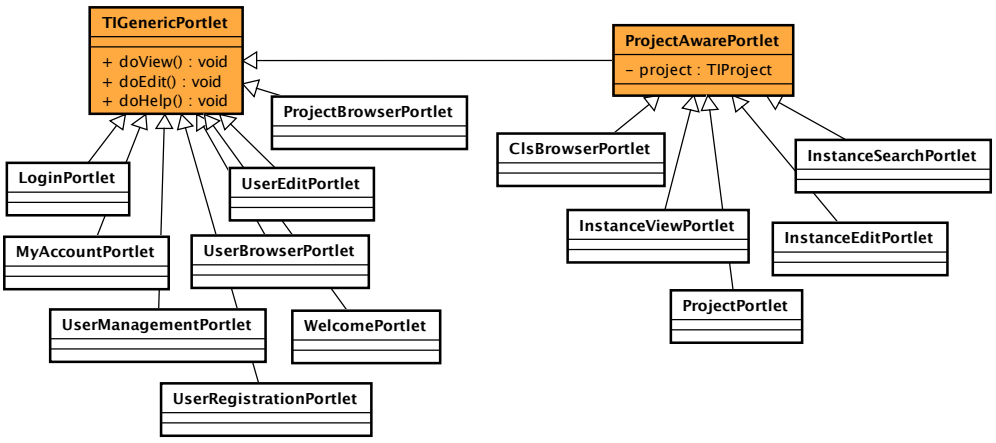

Abbildung 5: Klassendiagramm des Portletpakets

Dieses Paket enthält alle Portlets, die von teamInstance benötigt werden. Das Mehrbenutzerdesign unserer Applikation setzt dabei auf intensive Portletkommunikation; zur Implementierung der Kommunikation zwischen Portlets programmieren wir deshalb gegen die Portlet-Messaging-API in Version 2.2 (siehe Abbildung 6, Seite 9 für ein Beispiel des Ablaufs einer solchen Kommunikation). Bei allen Portlets gliedert sich der Lebenszyklus in die Erzeugung und Initialisierung, die Verabreitung der eingehenden Requests und der Beendigung und Entfernung. Dazu muss die Schnittstelle von `GenericPortlet` implementiert werden. Die JSPs verwenden die Portlet-Tag-Libary (Teil des Java-Portletstandards) für den Zugriff auf Portletelemente und Funktionen. Sie bilden die verschiedenen View-Komponenten unser Applikation, während die Portletklassen die Aufgaben des Controllers übernehmen (siehe dazu Abschnitt 3.5.1).

4.4 de.teaminstance.security

Dieses Paket enthält die Klassen zur Benutzerverwaltung. Dabei bildet die Klasse `TIUser` die Eigenschaften und Methoden eines teamInstance-Benutzers ab. `TIUserManager` und `UserRegistrationManager` beinhalten Funktionalitäten zum Verwalten der Benutzer. Die

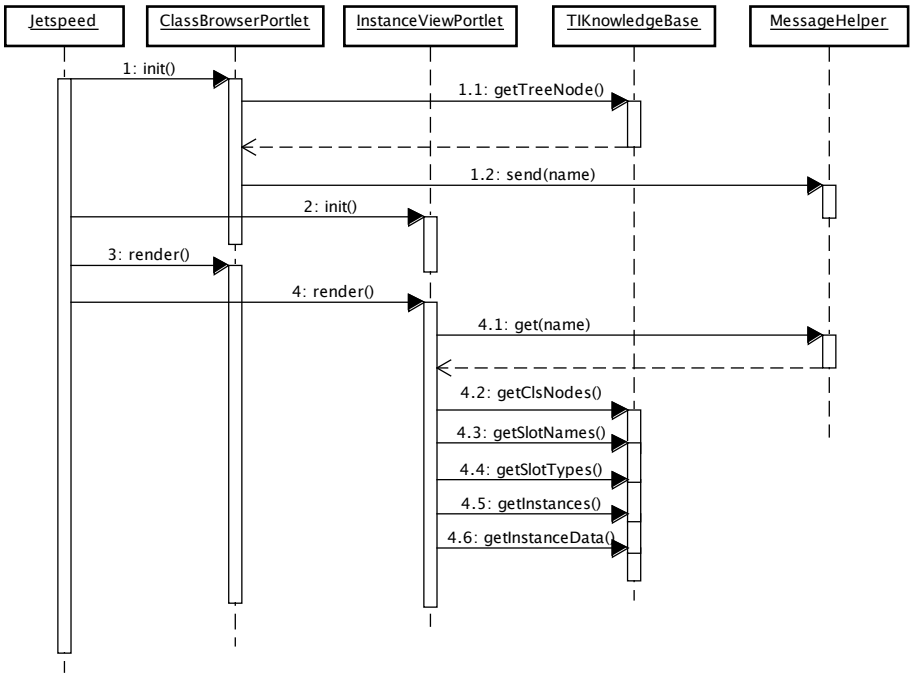

Abbildung 6: ClassBrowserPortelt und InstanceViewPortlet

Interfaces `TIUser`, `TIUserManager` und `UserRegistrationManager` dienen der Abstraktion vom verwendeten Portal, damit eine eventuelle Portierung möglich wird. Die Portalspezifischen Implementierungen der Interfaces befinden sich in den Klassen `TIUserImpl`, `TIUserManagerJetspeedImpl` und `UserRegistrationManagerJetspeedImpl`.

4.5 de.teaminstance.util

Im Paket de.teaminstance.util befinden sich Hilfsklassen zur Unterstützung der Funktionalität von teamInstance. So stellt `DataNode` einen Knoten eines beliebigen Baumes dar und kann im data-Attribut ein beliebiges Objekt speichern. Es kann verwendet werden um Klassenstrukturen einer Ontologie zu speichern oder den Verzeichnisbaum eines hierarchischen Dateisystems abzubilden. Weiterhin enthält de.teaminstance.util noch die Klasse `MailUtil`, welche Methoden zum Versenden von E-Mails kapselt.