Unified ModelingLanguage (UML)Based Design

Judiciary Information System (JIS)

119CS0100	POKALA KUSAL	GROUP 4
119CS0101	PRIYANSHU KUMAR	
119CS0102	SUSHREE SATARUPA	
119CS0103	NITIN AGARWAL	

Table of contents

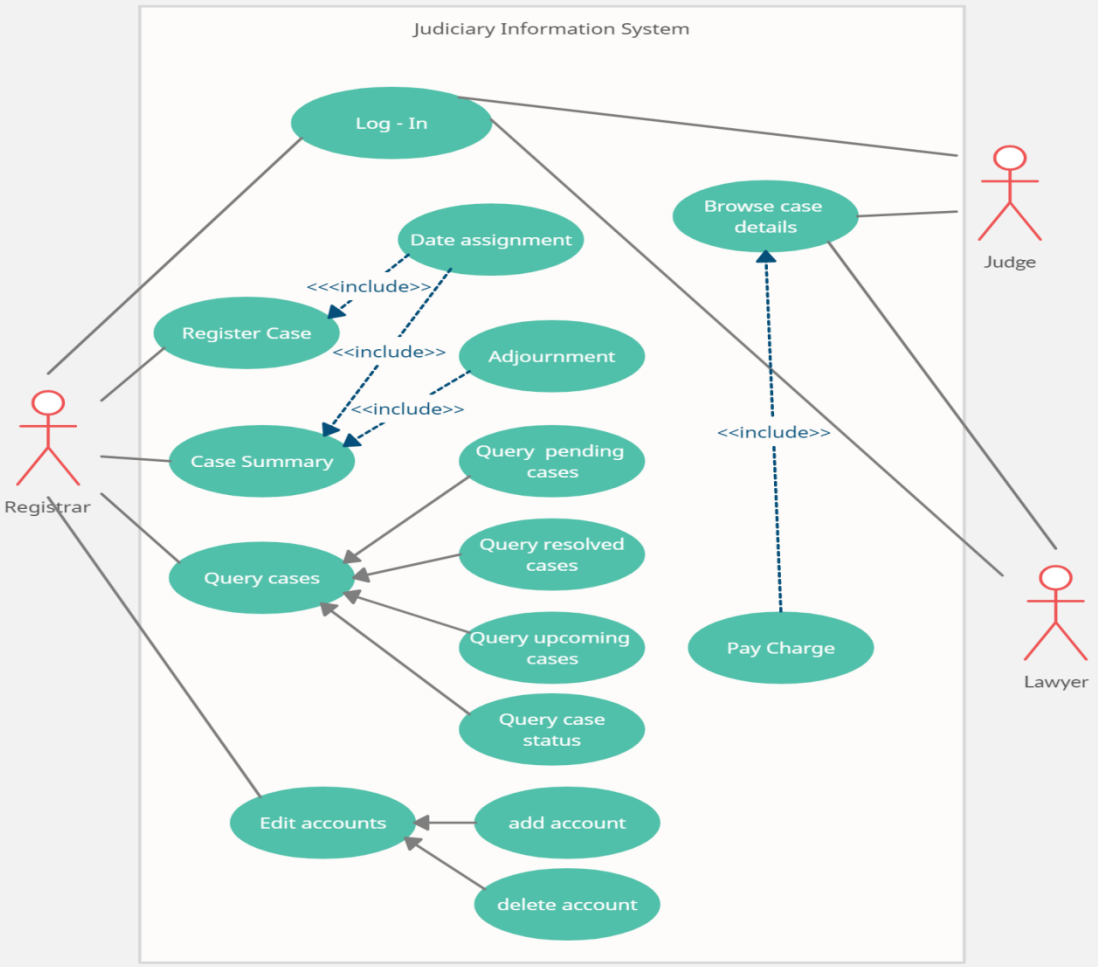
1. Introduction

This document builds upon the Software Requirements Specification and the SA/SD documents to detail a much more concrete view of the JIS to be implemented. The UML diagrams presented below have been drawn using **Umbrello**, an open-source tool available for this purpose.

NOTE: All the notations are part of the standard UML, thus they have not been documented separately.

2. Use case diagram

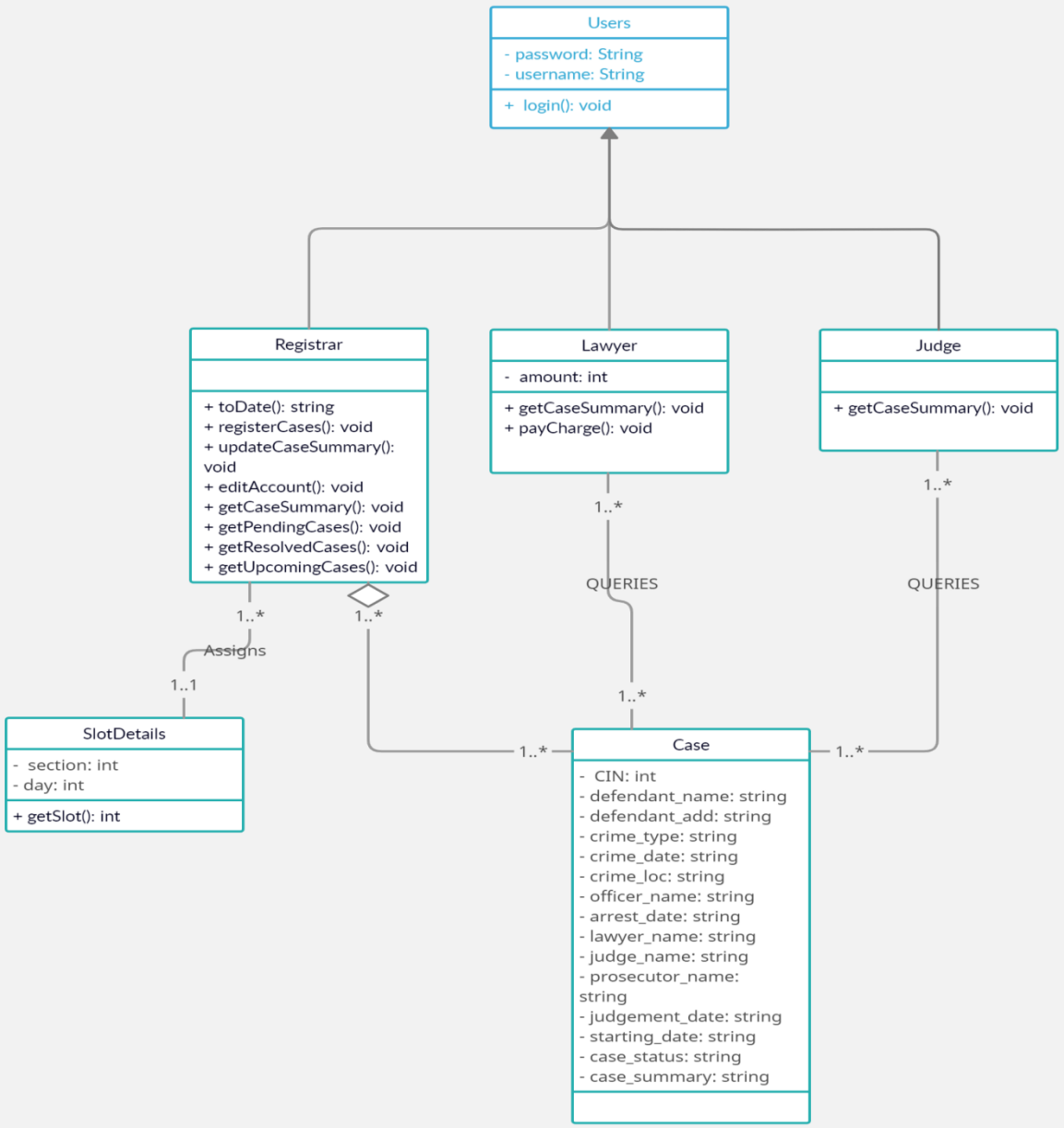
Brief description: All the three users of the system have a common use case for browsing existing cases. Apart from this, the registrar has many special use cases as shown below. Particularly, the 'close case' use case is an extension of the 'update case' use case because the registrar must have first initiated the latter use case to invoke the former. Further, the use cases for creation and deletion of user accounts are generalized versions of creation and deletion of specialized users, i.e. lawyers and judges. This relationship has been modeled appropriately.



3. Class diagrams

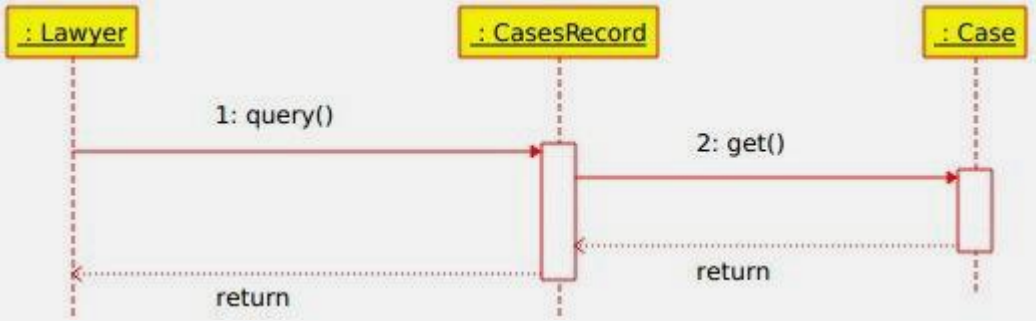
Brief description: User is a base class for all user types. Three classes are derived from the User class for the three stakeholders of the JIS. All the User objects are wrapped in an invariant container object of UsersRecord class, of which exactly instance is present in the system at all times. It contains functions for creating and deleting users. As marked in the diagram, there may be only one object of Registrar class.

All the details to be maintained for a court case are modeled as data members in the Case class. Further, objects of Adjournment and Hearing classes are used to model every update of the case. Analogous to the UsersRecord class, the CasesRecord class provides an invariant container object to wrap all the Case objects maintained in the system.



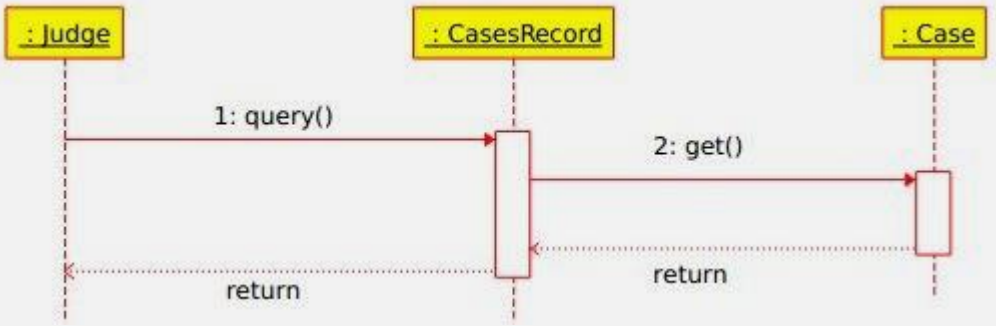
4. Sequence diagrams

4.1. Lawyer



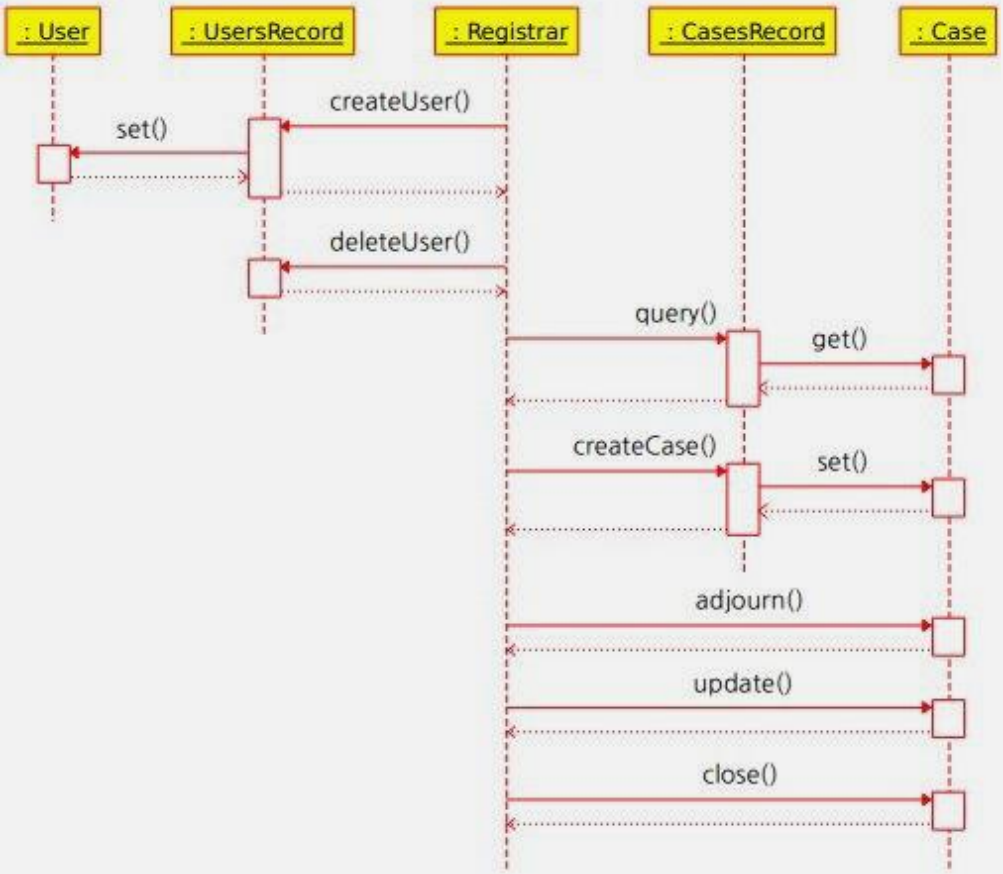
Brief description: Lawyer has only one function, i.e. browsing cases. For this purpose he sends a query containing keywords to the CasesRecord object in the system. This object then searches the data fields of all the cases to compile a list of matching cases. Here get() denotes getter functions for the various case data fields.

4.2. Judge



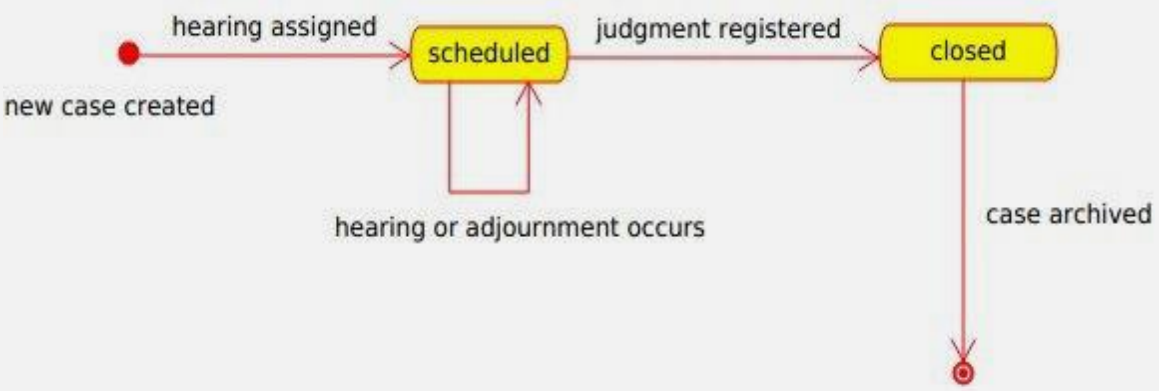
Brief description: As in the case of Lawyer, Judge also has only one function, i.e. browsing cases. For this purpose he sends a query containing keywords to the CasesRecord object in the system. This object then searches the data fields of all the cases to compile a list of matching cases. Here get() denotes getter functions for the various case data fields.

4.3. Registrar



Brief description: The user management and case management capabilities of the registrar are shown on two different sides of the diagram. While querying and creating cases the registrar needs to interact with the CasesRecord object whereas for other functions like adjourn, update and close, he can directly call methods of the Case objects. The get() and set() methods represent various getter and setter functions.

5. State chart diagram



Brief description: The above is the state chart diagram for a *case* object. When a new case is registered, its details are entered by the Registrar and a date of hearing is assigned, at which stage the case attains the 'scheduled' state. From here it continues to be in this state through all further hearings and adjournments. When a judgment is finally registered, it moves to the 'closed' state permanently. Thereafter it is archived in the system and ceases to feature in the currently scheduled (active) cases.