Automated Workflow System (AWS)

Software Design Document

Version 1.0.0 Status: draft

Prepared by Abhineet Pandey, Divesh Dodeja, Karan Sehgal, Vineet Madan
Team Members

IIT Ropar

12th February 2020

TABLE OF CONTENTS

1. Introduction

1.1. Purpose

The purpose of this document is to show and explain the design and the relevant decisions regarding the same in the context of Automated Workflow System (AWS) software.

1.2. Scope

The main purpose of AWS is to help businesses create customizable workflows i.e. series of processing steps to apply on certain data form(s). This system would not only help in increasing the efficiency of basic business workflows but also increase the productivity of the employees of the corporation by creating an interactive and secure interface to categorize and keep track of multiple business workflows. For a business, the product would provide the authorized personnel to create custom data forms required for a specific workflow. The personnel would also be provided with a detailed view of workflows and the activities of users within the workflow, thereby creating a transparent system [1].

1.3. Definitions, Acronyms, Abbreviations

DB	Database
ER	Entity Relationship
SRS	Software Requirements Specification
Admin	Administrator
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol *Secured*
SQL	Structured Query Language
QA Team	Question Answering Team
API	Application Programming Interface
JSON	JavaScript Object Notation
CSS	Cascading Style Sheets
HTML	HyperText Markup Language

JS	JavaScript
CSRF	Cross-Site Request Forgery

1.4. References

[1] Abhineet, Divesh, Karan and Vineet, *SRS-Automated Workflow System* Version 1.0, January 29,2020.
[2] Ullman, Garcia-Molina and Widom, *Database Systems, The Complete Book.* Pearson, 2009.
[3] Occam's Razor, Wikipedia, https://en.wikipedia.org/wiki/Occam%27s_razor (Accessed Feb 12,2020)

2. Design Overview

2.1. Description of Problem

The product is a replacement for traditional offline business workflow practices, where employees are required to process and forward data forms to respective business personnel in-person or through an intermediary. The product would provide a business with interactive web-based tools to automate the previously stated process accompanied by monitoring tools to make the process secure, transparent and convenient.[1]

2.2. Operating Environment

The initial version of the software should be deployed on a Linux machine with both SQL and NoSQL databases installed. A python and node development environment should also be available with permissions for installing open-source modules and internet access.

2.3. Technologies Used

2.3.1. Hardware Interfaces

Since the web portal does not have any designated hardware, it does not have any direct hardware interfaces, except the hardware the web portal would be accessed through, like a computer or a mobile device. [1]

2.3.2. Software Interfaces

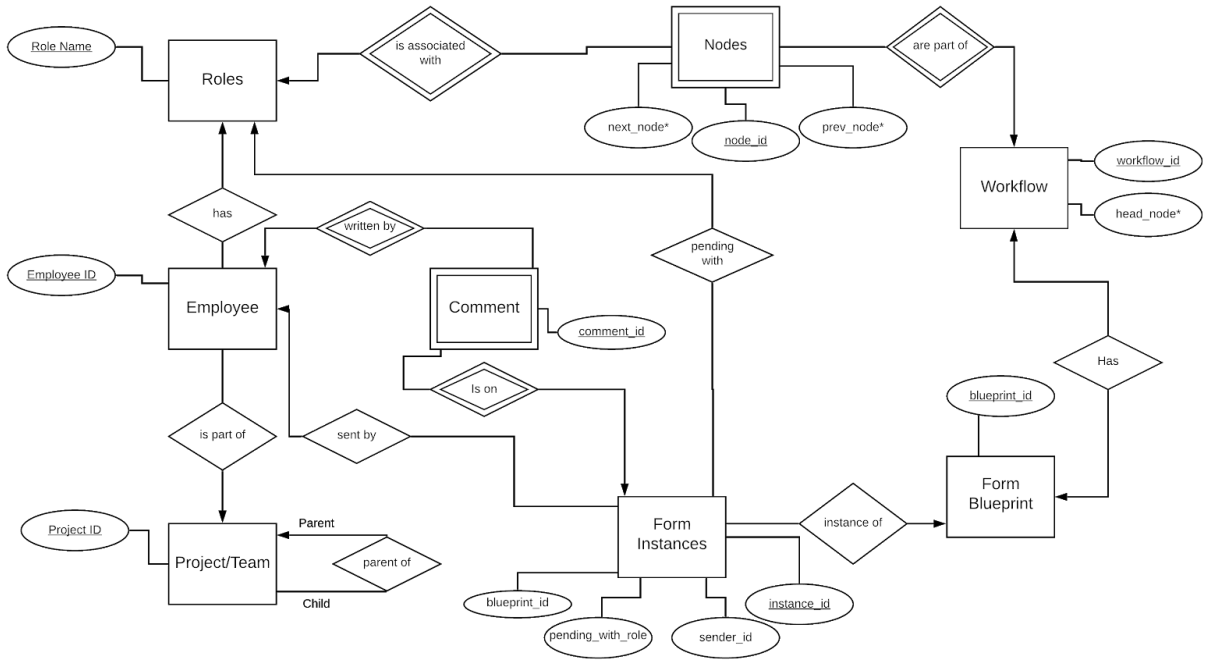
The application communicates with both the SQL and NoSQL based databases in order to add, retrieve, update and delete relevant information. The user-interface will be built using HTML, CSS and JavaScript and its libraries. The product will be developed using Django, a python based web framework. The standard data exchange format will be JSON.

2.3.3. Communication Interfaces

Communication between different parts of the system is through HTTP or HTTPS protocols.

3. Design Artifacts

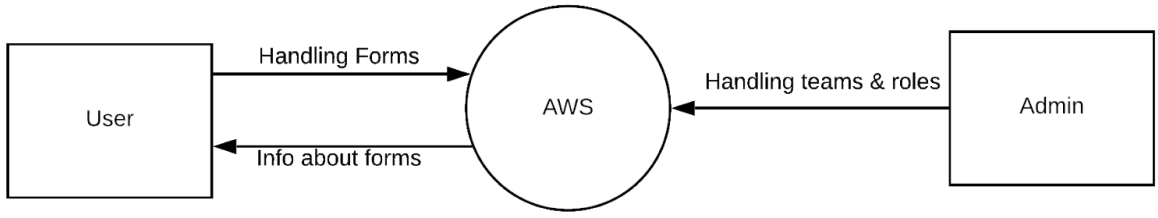
3.1 Domain Model Diagram



Entity-Relationship Diagram, convention followed as in [2].

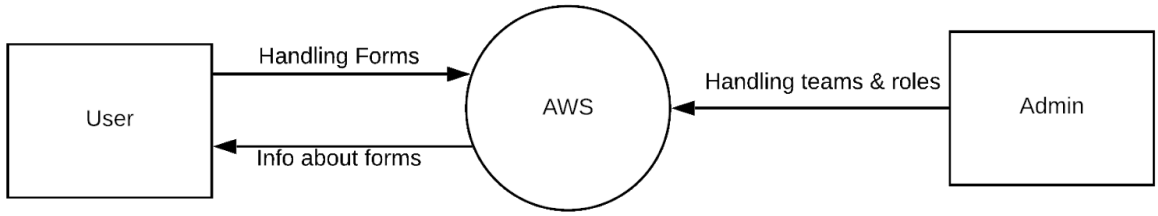
3.2. Context Diagram

Context Diagram, provided below, provides a very high level overview of the system.

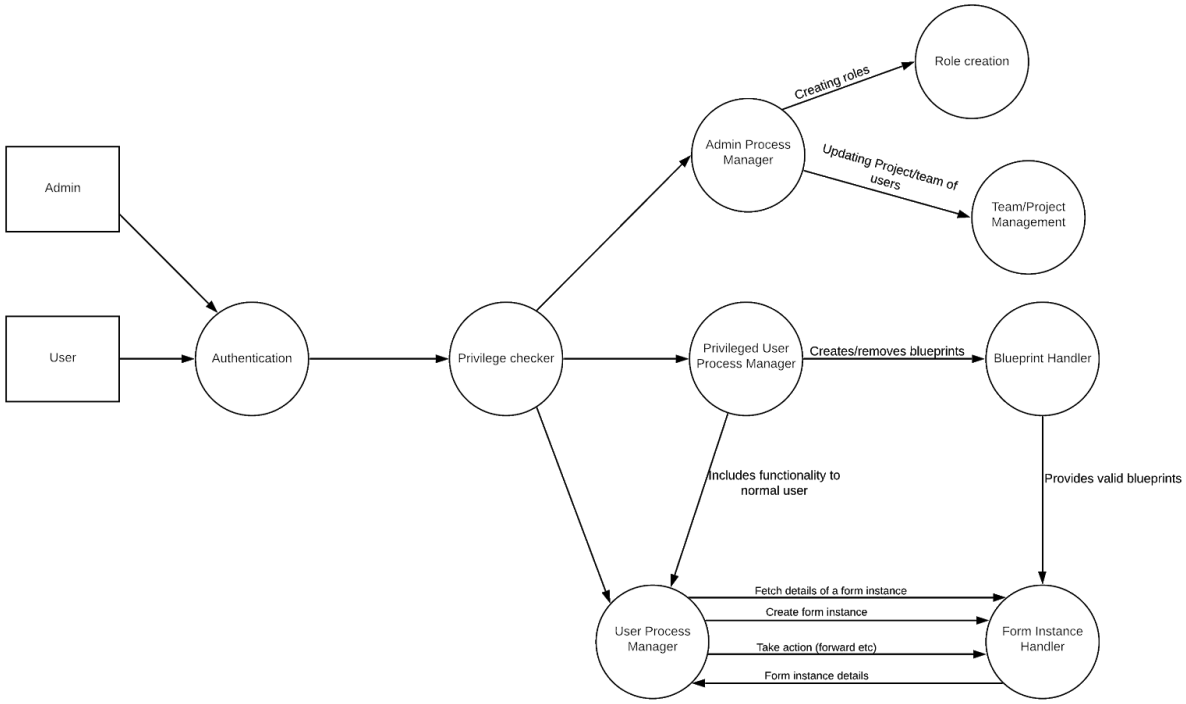


3.3. Data Flow Diagrams

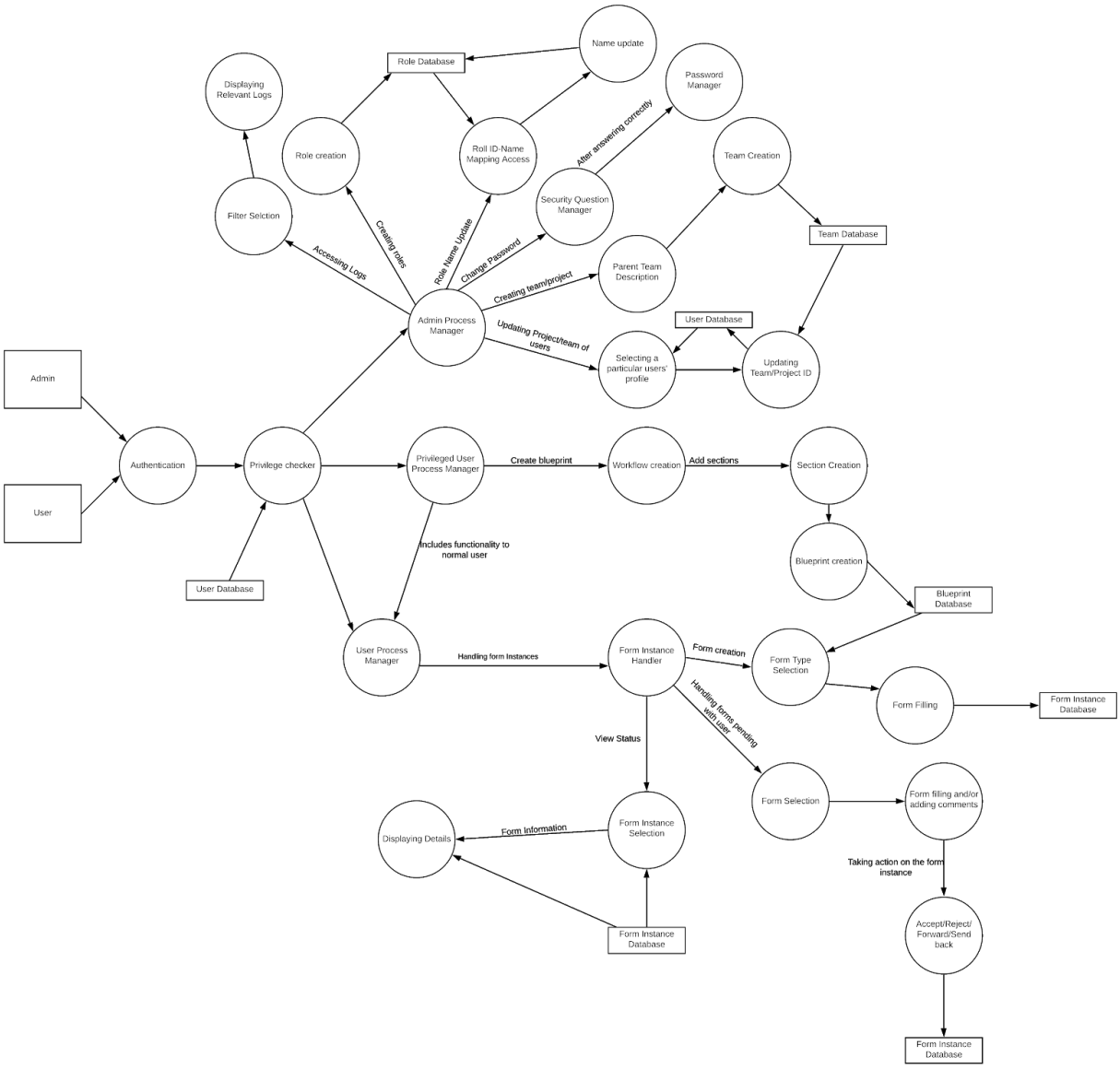
3.3.1. DFD-0



3.3.2. DFD-1

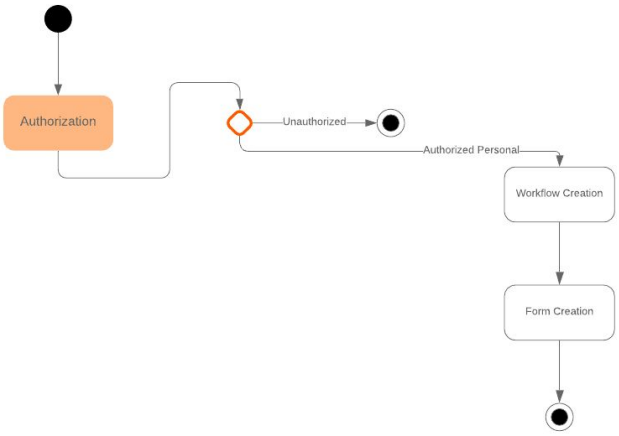


3.3.3 DFD-2

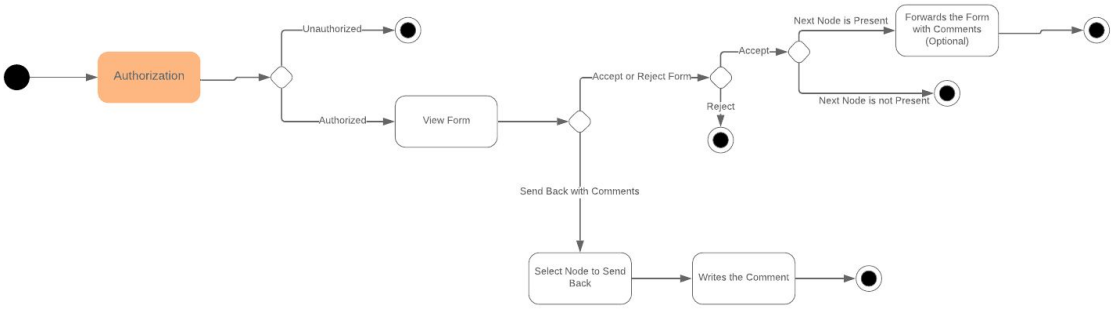


This diagram captures the use cases mentioned in the SRS[1].

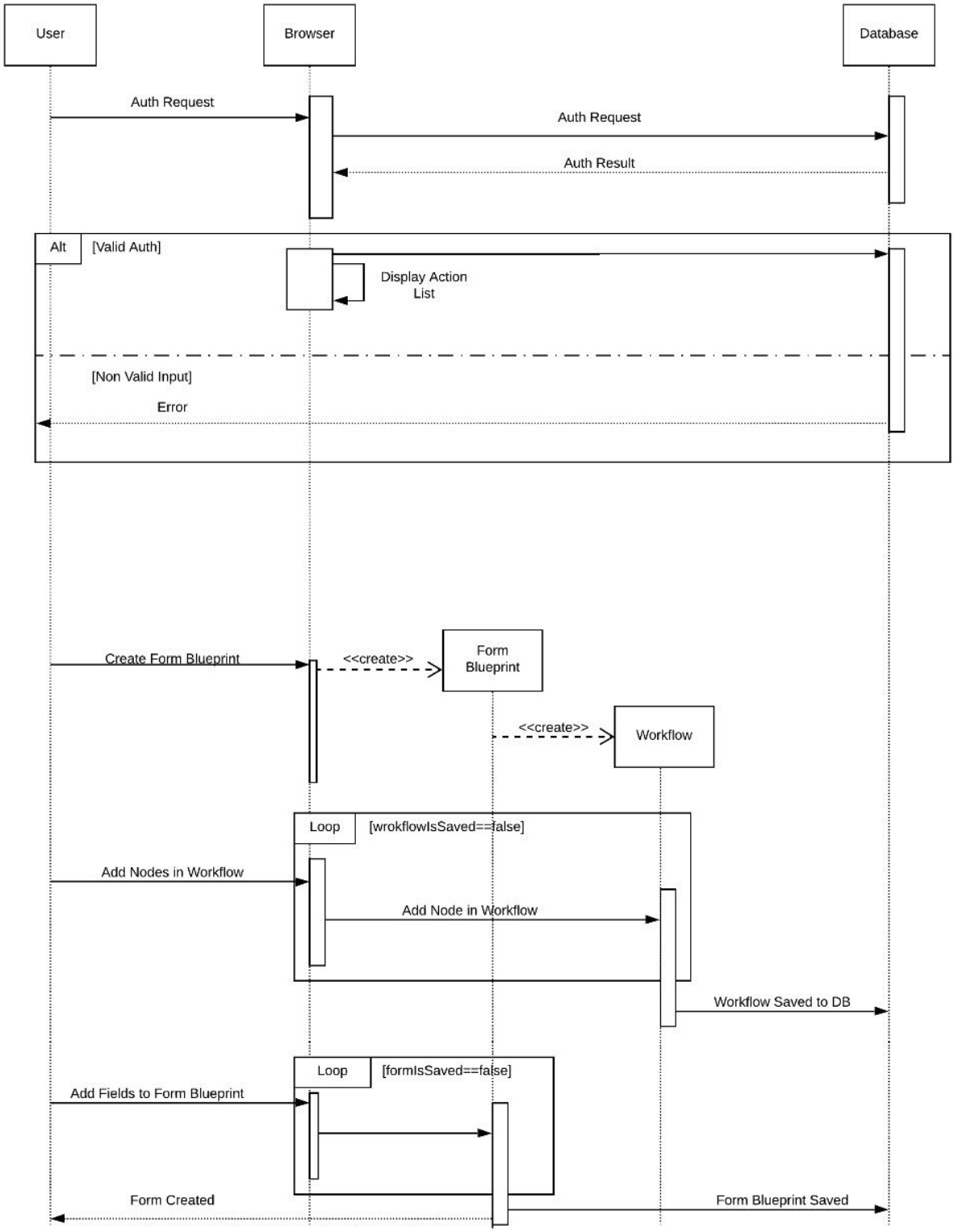
3.5. UML Diagrams for Dynamic of Static Structure



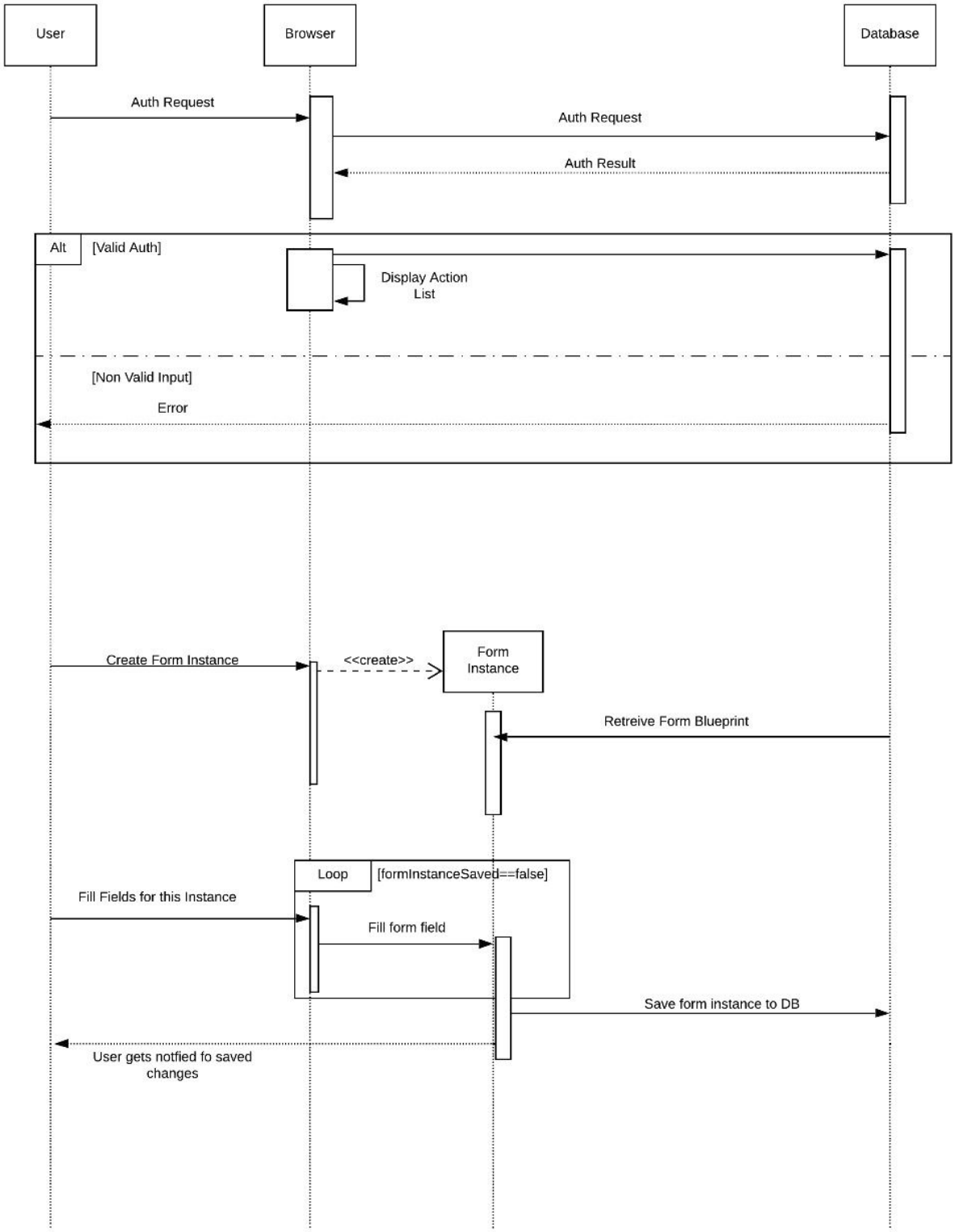
3.5.1 Activity Diagram: Form Creation



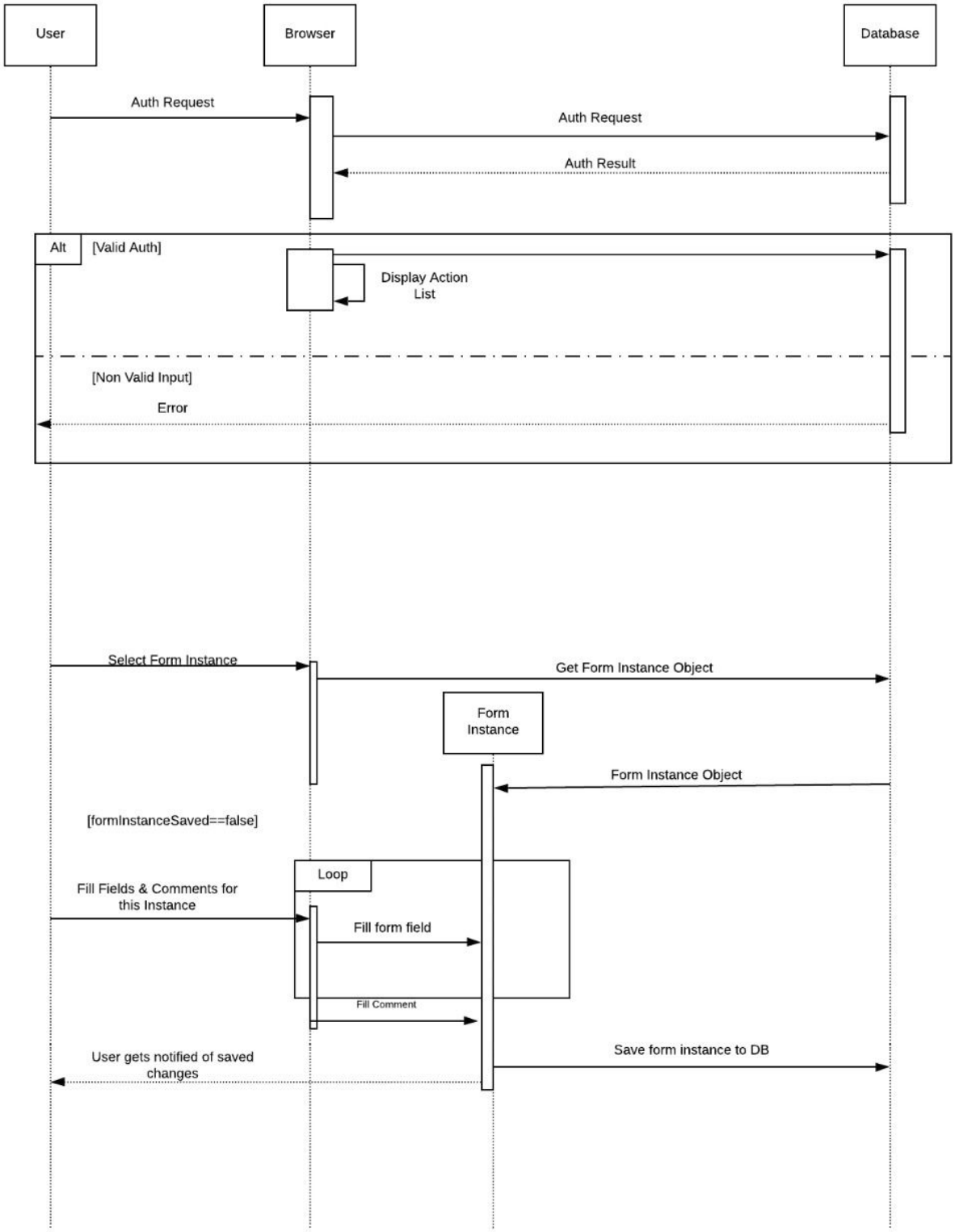
3.5.2 Activity Diagram: Filling form at an intermediate node



3.5.3 Sequence Diagram: Create form blueprint

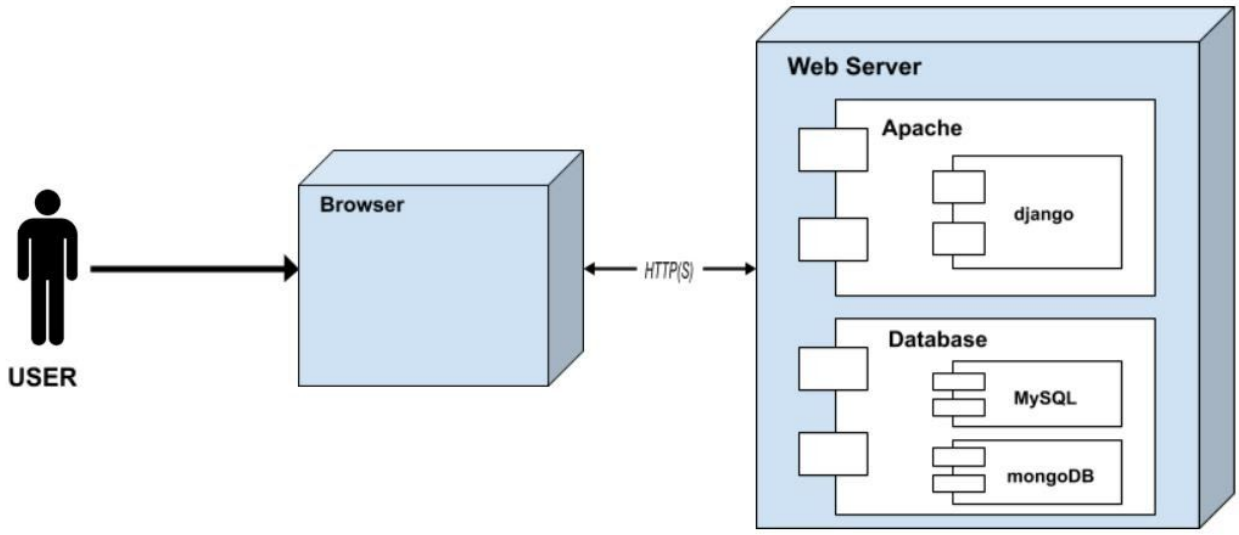


3.5.4 Sequence Diagram: Create Form Instance



3.5.5 Sequence Diagram: Filling Form (at intermediate node)

3.6. Deployment Diagram



4. Design Decisions

4.1 Whether to separate the front end and back end servers

Currently, we are having the same front end and back end servers, and hence, we would provide our service to clients who have their own server for databases and would run our application in their intra or inter-net. This would give them more security on their sensitive data and would increase their customizability (since they would have access to the source code as well).
As there is no middle ground between frontend and backend, this reduces the risk of man in the middle type of attacks. However, the backend becomes susceptible to risk if the frontend server is compromised.

4.2: Regarding security features

The passwords are stored in an encrypted manner. A miniscule time is taken while authentication for conversion to hashcode and matching, but this cost can be paid as it makes the system much more secure, and the risk of exposure of passwords in plain text to an attacker reduces. Each POST request in the system shall be accompanied by a corresponding CSRF token as a request parameter. This is done to ensure that a valid request can not be generated by an attacker. [1]

4.3: Regarding the database model or ERD diagram

The major design decisions that can be interpreted from the ERD diagram are discussed hereon.
The form blueprint is a separate entity, which has associated workflow, defining the roles the particular form has to go through in the process of approval. Workflow has an associated weak entity, nodes, which define the units of the workflow, i.e. the roles associated with each step in the workflow.
As comments as such are not a feature in a blueprint, but rather in a form instance, they have been associated with form instances as a weak entity.
The ERD also clarifies that each employee is part of some project/team, this ensures that no ambiguity remains in case a form is supposed to the reporting manager of the form sender, as the reporting manager can be identified using the project/team of the sender of the form.
Foreign key constraints are put wherever required to ensure referential integrity and prevent the possible insertion of non-conforming data.

4.4: Form blueprint once created cannot be edited or deleted, only marked as inactive.

Since a blueprint would have some corresponding data, editing or deleting the blueprint would corrupt the said data. So we have made the design choice that once a blueprint is created it cannot be edited or deleted, since that would corrupt the relationships between entities already existing in the data.
If the user really wants to go for editing the form blueprint, he/she has to create a new workflow.
Apart from this, we have given a deletion like feature where the blueprint can be marked as inactive and won't show up in the users' interface.
The downside of this decision is the presence of irrelevant form blueprints in the database. However considering the fact that the number of blueprints in an organisation are limited, it doesn't make a heavy impact on search performance or the memory occupancy.

4.5: Role once created cannot be deleted, it can only be marked inactive

Please refer to the section 4.4. The same argument applies here and a role once created cannot be deleted, only marked inactive.

4.6: Form has sections and each section has a corresponding node which can fill or edit the section.

A form blueprint while creation has certain sections. Say, a form for leave application in an academic institute has sections pertaining to cross-cutting faculty like Dean, HOD etc, in other words, apart from the sender of the form, it may so happen that people from other certain roles have some sections to fill in the sender's form. In this document, we refer to a role associated with a particular section as a node.
The need for this functionality arises in the cases where we want users with only certain roles to modify a particular section of the form. This feature is supported by the role assignment and authorization features of the system.

4.7: Comments are visible to all and are not "tagged"

Apart from sections in a particular form instance, additional comments can be added on the form by the user the form is pending against. These comments are visible to everyone who has access to the form. The comments are not 'tagged', in the sense that they are not 'assigned' to a particular person in the workflow. We leave it to the users in the workflow to interpret as to who is supposed to reply to comment, in case a reply is required. As the form, after adding a comment, can be sent back in the workflow, we believe it is intuitive as to who is supposed to respond to a particular comment and it can explicitly written in the comment as well, in case the reply is required from a particular user.
The visible con is that the comment may be directed towards a particular user and hence they are the one who should be replying to that comment, however, including such feature increases the complexity of the software, without any specific necessity as the idea of tagged is user is implicit and intuitive even in the software without this feature. We prefer to use Occam's razor[3] here.

4.8: Use of SQL and NoSQL databases

Since entities like user, team, etc share a relationship, we use a relational database like SQL to manage them. Appropriate foreign key relations have been defined in order to ensure referential integrity.
However, the entity form(form blueprint and instance), apart from its relation to other entities, is unstructured in its attributes (since a specific blueprint can have different kinds of data fields, and each blueprint can be structurally different from another). Hence, we should use a NoSQL database to manage them.

4.9 Notion of teams and relation between them.

An entity called team has been created, which can also be semantically inferred as project, however, for the sake of a well-defined hierarchy in terms of workflow, a person can not be a part of more than one team at once. In an organisation, it may so happen that groups with same or similar hierarchy exist, the notion of team is designed to capture that idea.
We have designed parent-child relations between teams. These relations become useful in the case where same or similar hierarchy is present within two teams, and inter-team hierarchy is also present.

4.10 Logging

Each event involving change in state of any entity will be logged. All log-ins shall also be logged. The logging will be done in a manner that includes appropriate flags and follow some loose format, so that accessing the logs becomes easier by using filters, which can be implemented easily in the scenario where suitable flags are present in the logs along with the statements.
Exceptions thrown by the application would also be logged in a file, so it becomes easier to keep track and debug the exceptions, while the user whose interaction caused the exception would be informed in a manner in which information about the internal workings is hidden (e.g. a simple modal saying something went wrong) and redirected accordingly.

4.11 Non-Functional Requirements

Usage of an SQL based database for the majority of our application provides us with the ACID properties. We would be using an ORM to manage most of our functions, however, for certain complex scenarios, we would use native SQL to increase the performance of our application. Specific test cases would be designed in order to check the functionality of the application, such that code coverage is maximized.

4.12 UI/UX

The UI shall be designed in manner that information should be clustered into relevant groups, say type of form blueprints available. Such information may be displayed using techniques like drop-down menu, so that the screen doesn’t look cluttered and user experience doesn’t get hampered. Most frequently used options, say accessing pending forms, shall be easily accessible to the user, so that the usability of the application is high and minimal time is required for a user to understand the basic features of AWS.