

## LICENSE MANAGEMENT DESIGN DOCUMENT

Contents

[1. Technologies: 2](#_Toc105229918)

[a) Django 2](#_Toc105229919)

[b) HTML/CSS/Bootstrap: 2](#_Toc105229920)

[c) jQuery 2](#_Toc105229921)

[d) SQLite DB 3](#_Toc105229922)

[2. Deployment: 3](#_Toc105229923)

[a) Application Deployment 3](#_Toc105229924)

[b) SQLite DB 3](#_Toc105229925)

[3. Workflows (Snap shots): 3](#_Toc105229926)

[a) User Management: 3](#_Toc105229927)

[b) License Upload Flow 4](#_Toc105229928)

[c) License Approval Flow 6](#_Toc105229929)

[d) License Search Flow 7](#_Toc105229930)

[4. Deliverables 7](#_Toc105229931)

[5. Appendix 7](#_Toc105229932)

[a) Coding Standards: General coding standards for the Project is documented in this section. 7](#_Toc105229933)

[b) The Code layout: 7](#_Toc105229934)

[c) Imports, Blank Lines, and the Indentations: 7](#_Toc105229935)

[For Example: - 8](#_Toc105229936)

[d) The length of the Line and the Line Breaks 8](#_Toc105229937)

[For Example:- 8](#_Toc105229938)

[e) Whitespaces, Trailing Commas, and String Quotes 8](#_Toc105229939)

[For Example:- 9](#_Toc105229940)

[f) Naming Conventions 9](#_Toc105229941)

[For Example: - 9](#_Toc105229942)

[g) Exception Handling for every critical situation 9](#_Toc105229943)

[For Example:- 9](#_Toc105229944)

[h) Documentation of a Method 10](#_Toc105229945)

[For Example:- 10](#_Toc105229946)

# Technologies:

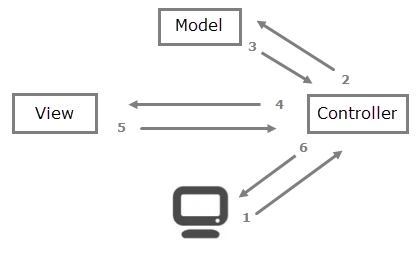
# Django

Django follows MVC pattern very closely, but it uses slightly different terminology. Django is essentially an MTV (Model-Template-View) framework. Django uses the term Templates for Views and Views for Controller. In other words, in Django views are called templates and controllers are called views. Hence our HTML code will be in templates and Python code will be in views and models.

**Models:** Models represents how data is organized in the database. In other words, in MVC pattern we use models to define our database tables as well as the relationships between them.

**Views:** A view is what you see when you visit a site. For example, a blog post, a contact form etc., are all examples of views. A View contains all the information that will be eventually sent to the client i.e., a web browser. Generally, views are HTML documents.

**Controllers:** Controller controls the flow of information. When you request a page that request is passed to the controller then it uses programmed logic to decide what information is needed to pull from the database and what information should it pass to the view. The controller is the heart of the MVC architecture because it acts as a glue between models and views.



# HTML/CSS/Bootstrap:

HTML is the standard markup language for Web pages. We can add components to create our page with HTML.

CSS is the language we use to style an HTML document. Bootstrap (a CSS Framework) is used to describe how HTML elements should be displayed

# jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

# SQLite DB

SQLite is a database engine written in the C language. It is not a standalone app; rather, it is a library that software developers embed in their apps. As such, it belongs to the family of embedded databases. SQLite is free to use database.

# Deployment:

This Section describes the Deployment and configuration for each of the Items in the Dev/ Prod Environments

# Application Deployment

1. URL: the URL where the Application could be downloaded.
2. Steps: The deployment and configuration steps are detailed below.
3. Configuration: The deployment and configuration steps are detailed below.

# SQLite DB

1. URL: the URL where the Application could be downloaded.
2. Steps: The deployment and configuration steps are detailed below.
3. Configuration: The deployment and configuration steps are detailed below.
4. Table Schemas and ER Diagram

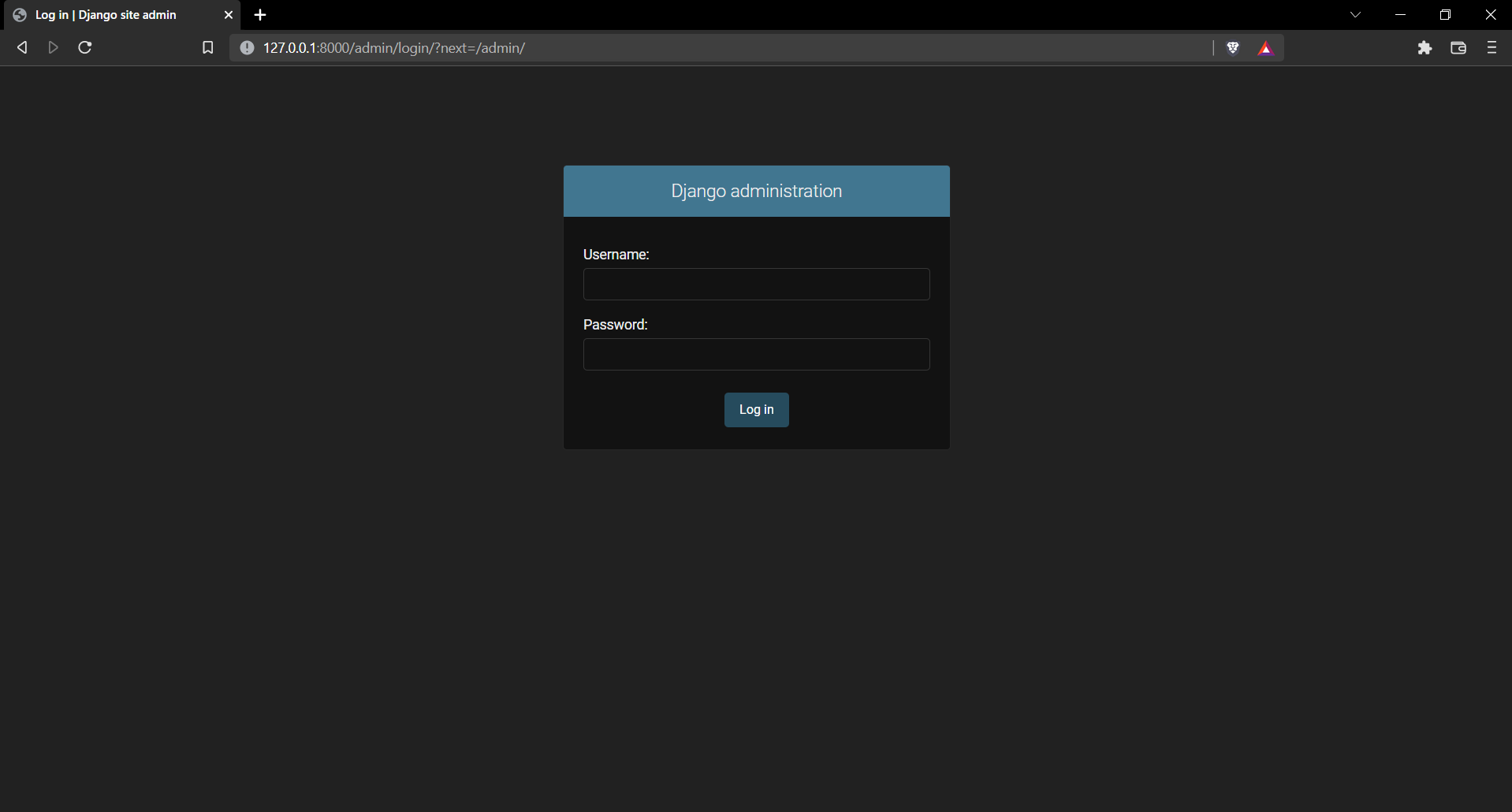
# Workflows (Snap shots):

# User Management:

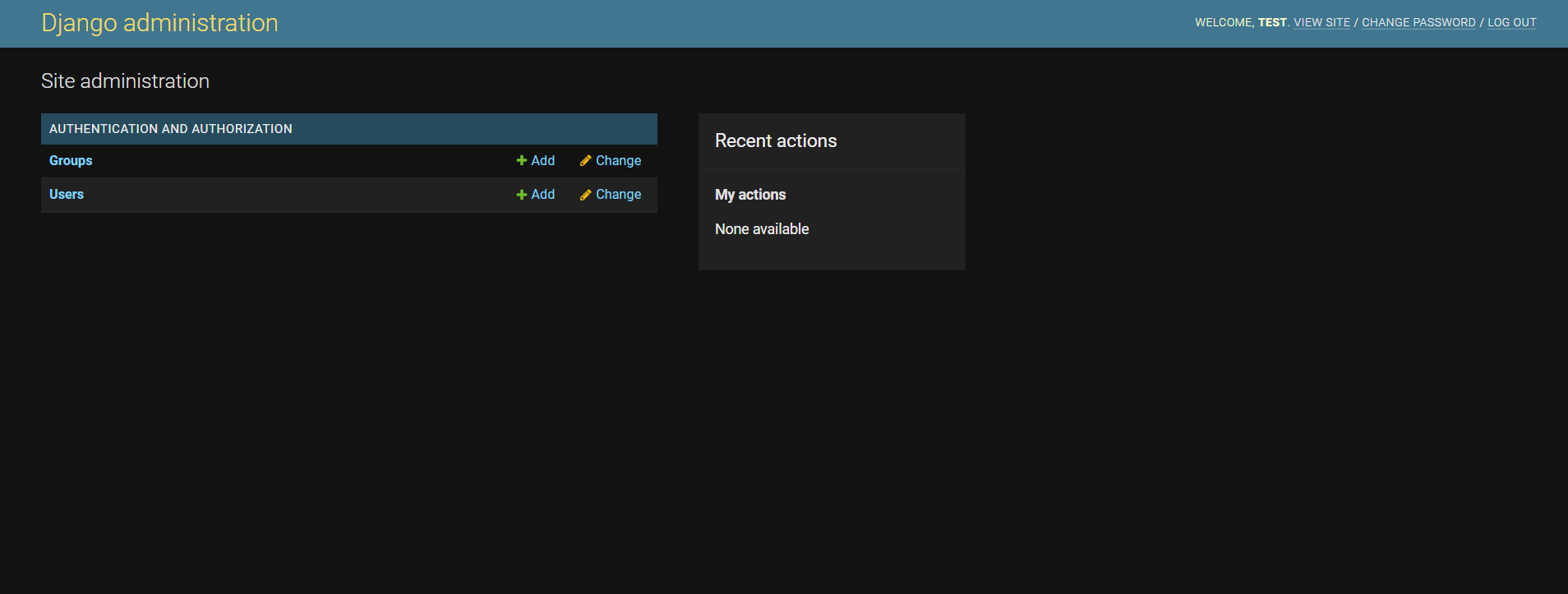
The User Management flow is explained in detail in this section with the Snapshots.

* User Creation
  + Role based
    - Normal User
    - Approval User
* Validations
* Screenshots

Django Admin Screen



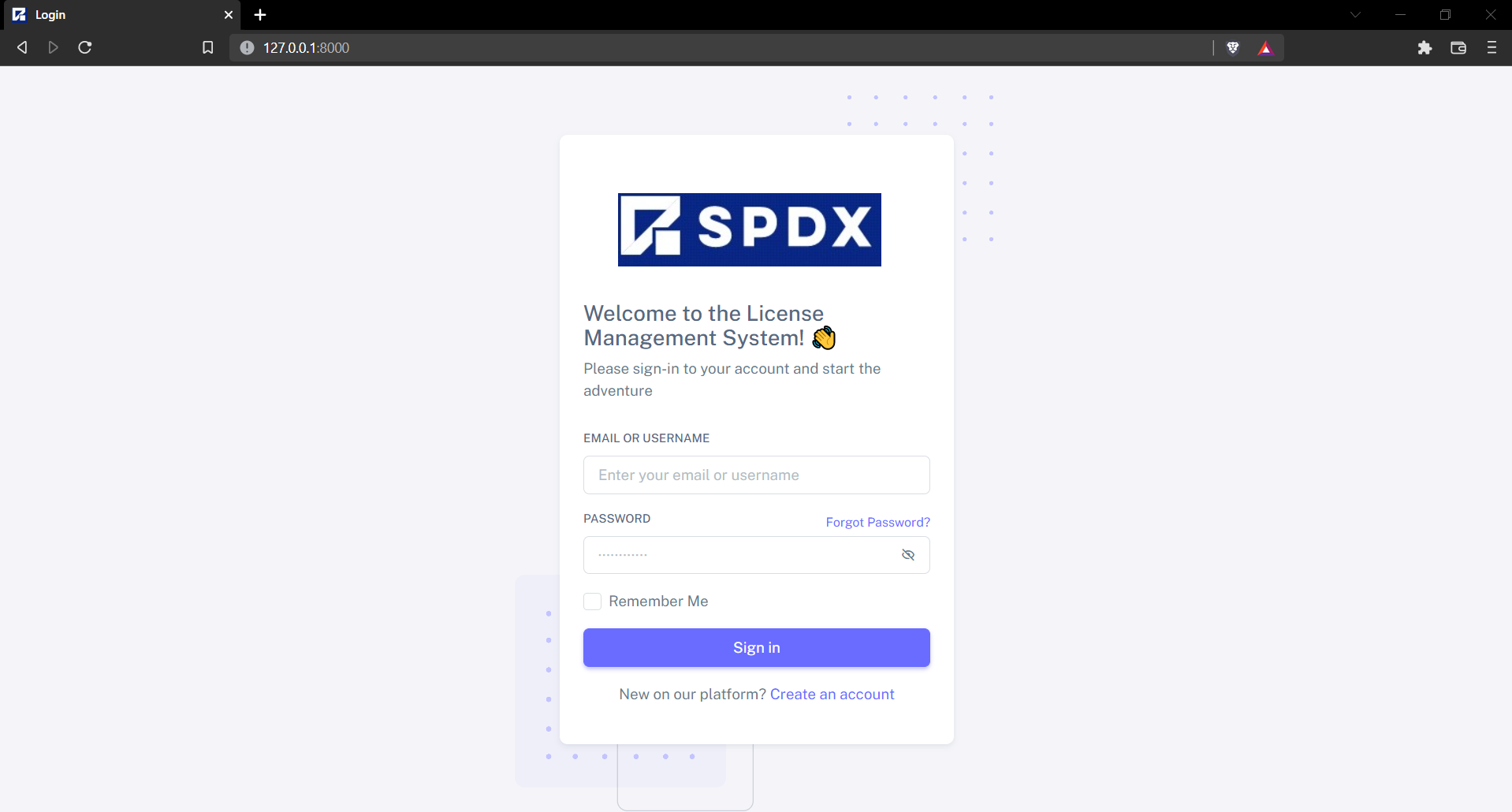
Django admin dashboard



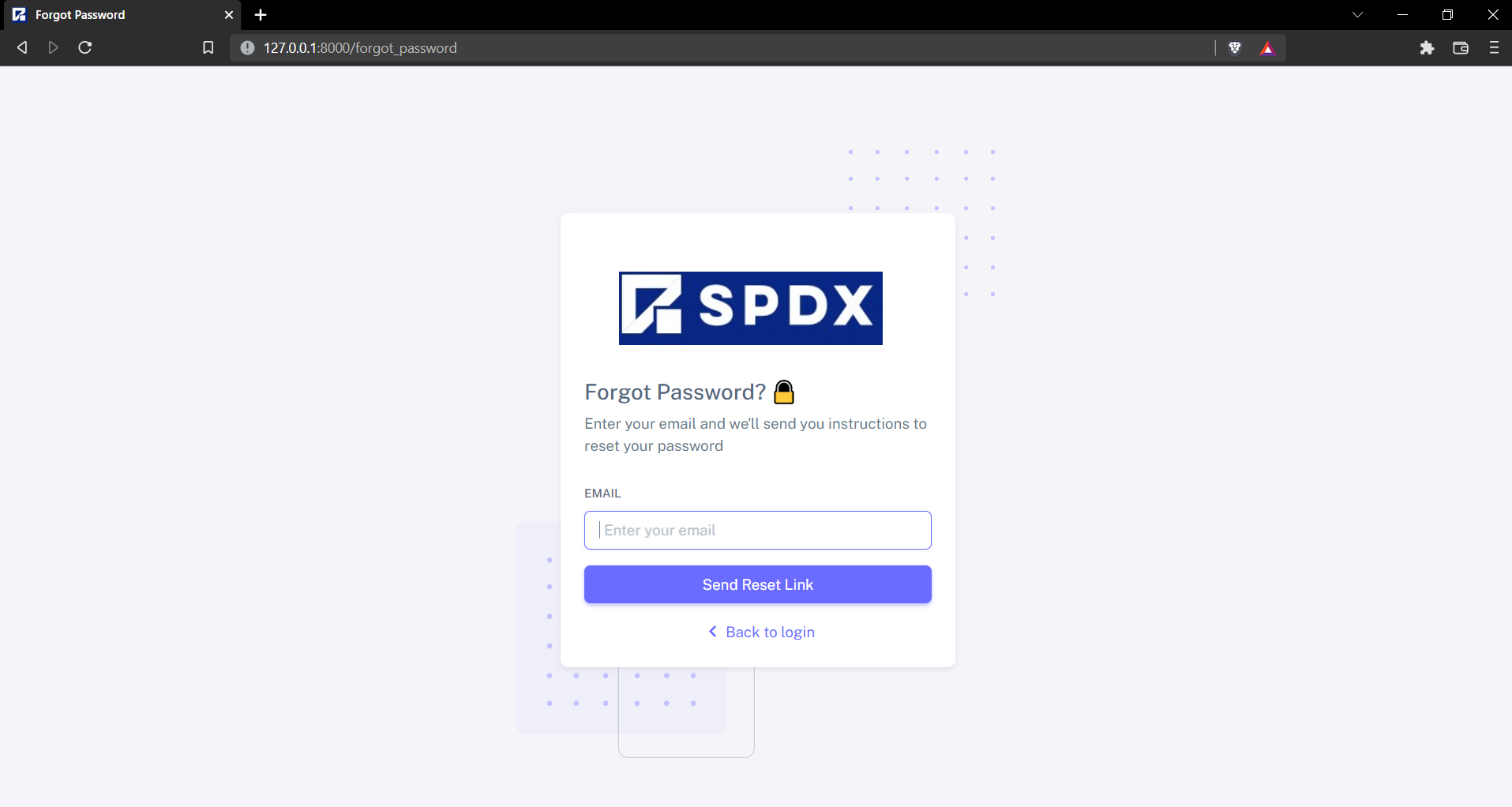
# License Upload Flow

* Session Management
* Update History of the License
* UI Models - MVC, for each screen to be Noted
* Database Model
* Validations
* Screenshots:

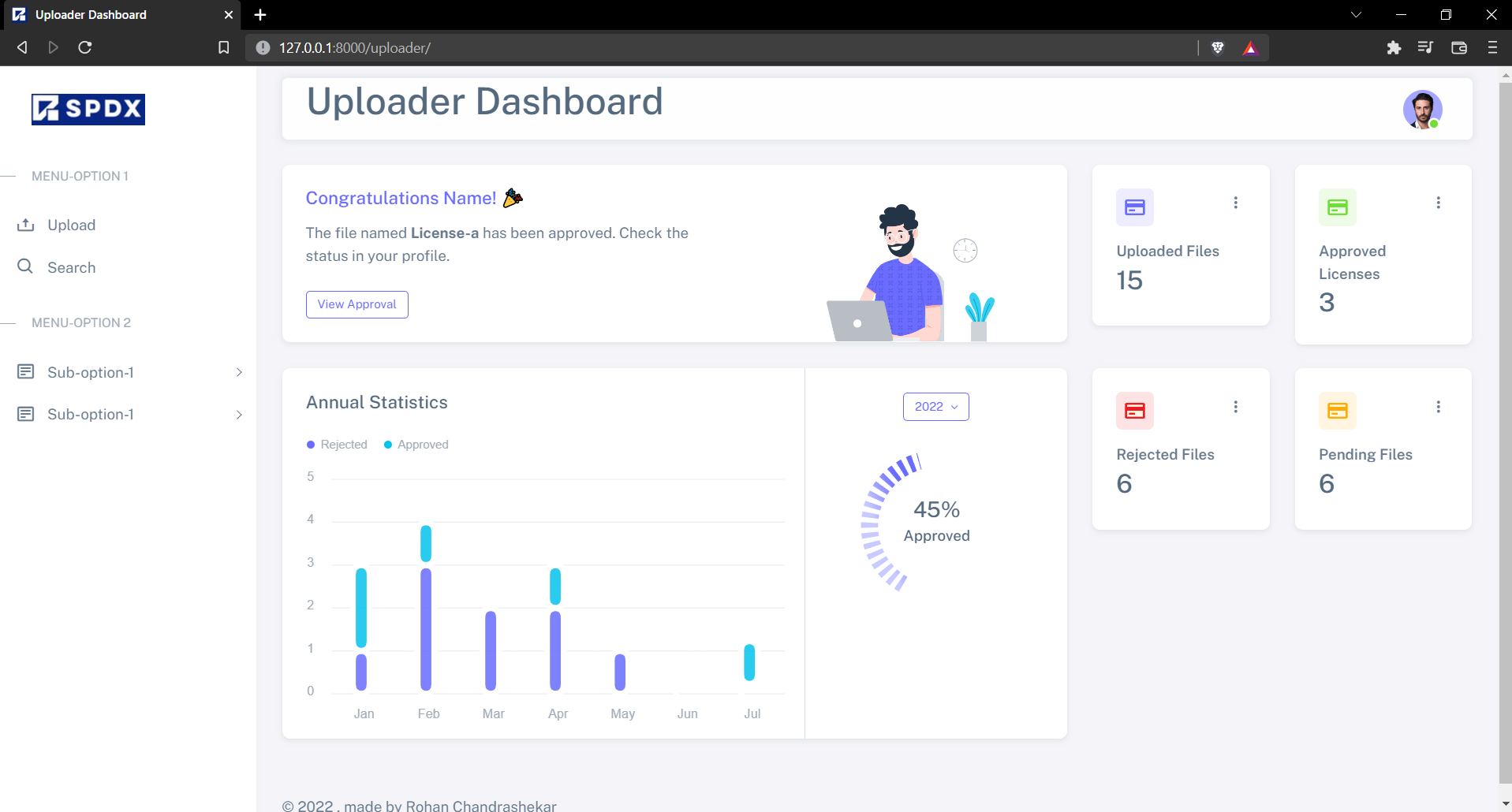
Application Log-in Screen



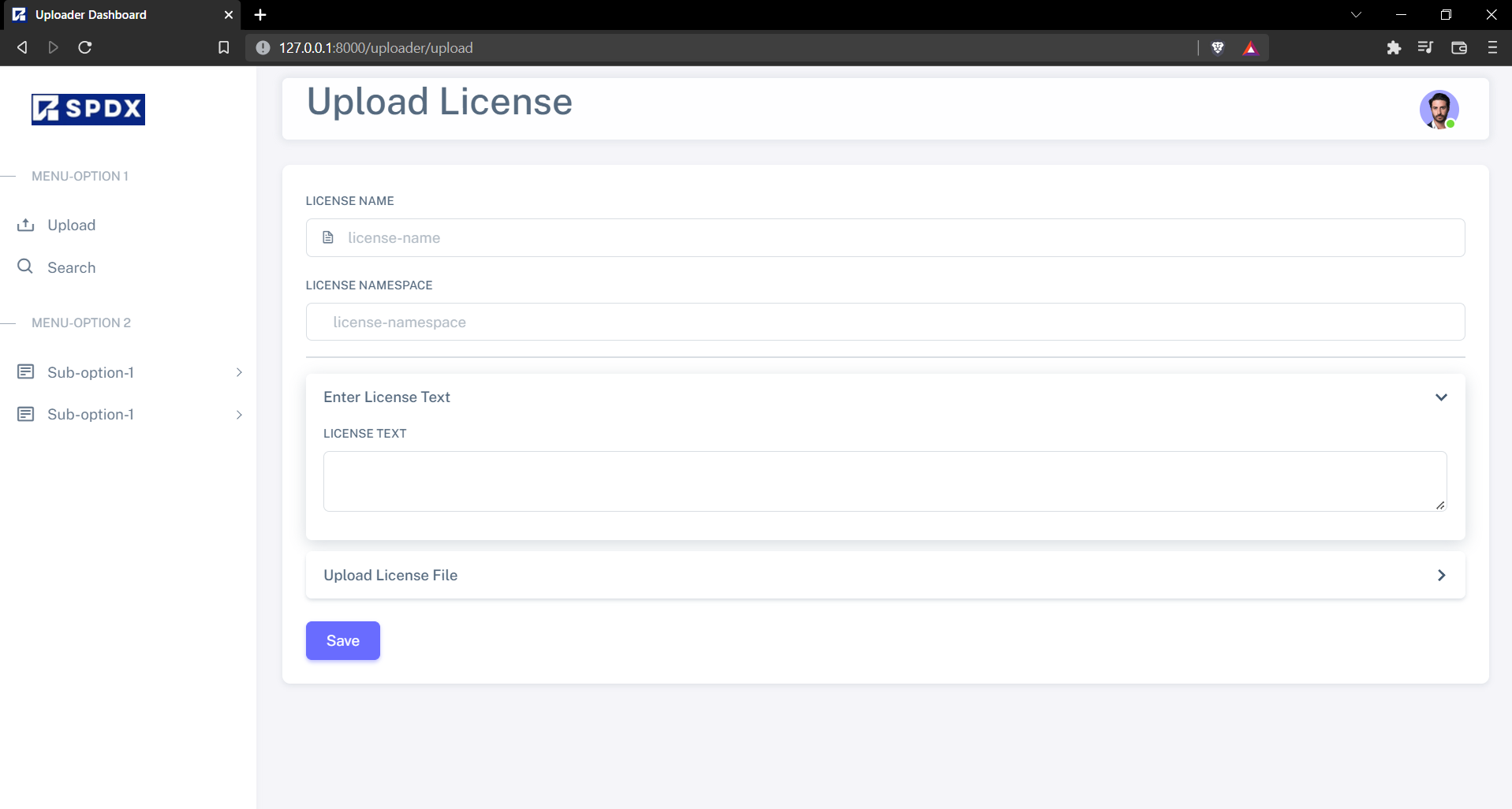
Forget password screen



Uploader Dashboard



License Upload Screen

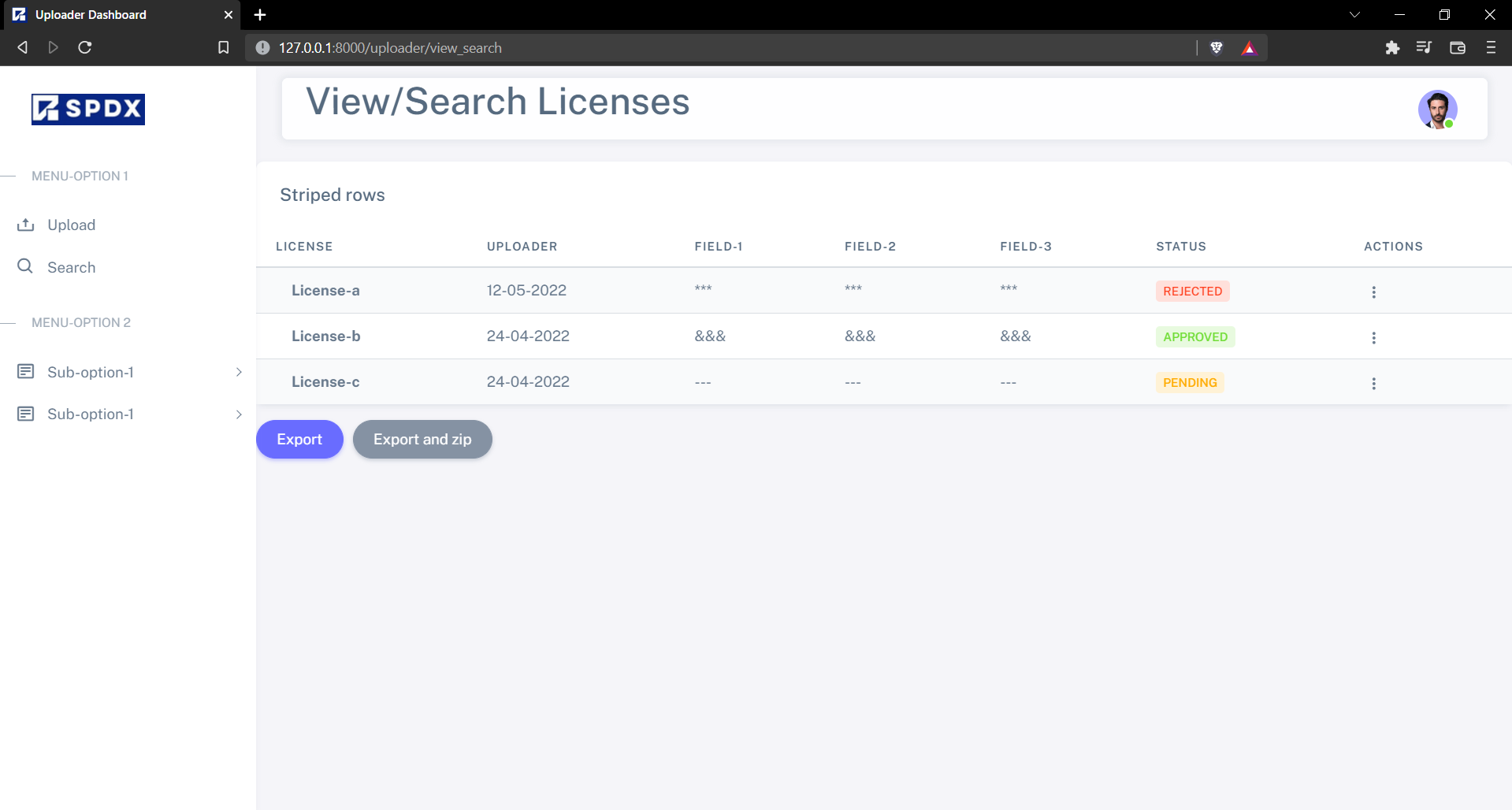


# c) License Approval Flow

* Session Management
* Update History of the License
* UI Models - MVC, for each screen to be Noted
* Database Model
* Validations

# d) License Search Flow

* Search Criteria
* Search Results
* Database Model
* Validations



# 4. Deliverables

The list of Deliverables for this project are: -

* Design document of the Project
* Installable for the Project
* Deployment document with detailed steps for each component
* User Manual, explaining in detail the admin tasks, uploader/ user tasks and Approver tasks.

# 5. Appendix

# Coding Standards: General coding standards for the Project is documented in this section.

# The Code layout:

This consists of Indentation, the maximum line length to be used, line breaks and blank lines, imports in python, and under names

# Imports, Blank Lines, and the Indentations:

The import should be in a particular sequence. At first, the standard libraries, then the third party, and at the last, the local libraries should be imported. If only a single function/class is needed from the import, I have done an absolute import. It makes the code much cleaner, accurate, and easy to identify. A space has been added between different types of imports.

There are two blank lines surrounding classes and top-level functions. The methods inside of the class is surrounded by a single blank line only. The preferred method of indentation is spaces.

## For Example: -

# first of all, the standard library imports

import standard\_library\_import\_a

import standard\_library\_import\_b

# then,the third party imports

import third\_party\_import\_a

import third\_party\_import\_b

import third\_party\_import\_c

# at the last, local library import

from local\_library import local\_a, local\_b

from local\_library\_two import local\_c

# two blank lines for top level functions

def top\_level\_function(argument):

# A standard four space indent

print(argument)

# The length of the Line and the Line Breaks

The length of the line is not greater than 79 characters. In the case of docstrings and comments where a block of text is large, it is limited to 72 characters. For long multiple case statements, the backslashes are also used. For using log statements with binary operators, python suggests breaking the formula line before the binary operator for better readability and this has been followed.

## For Example:-

def sample\_function(arg1, arg2):

'''

The document string length for a single line should be less than

72 characters. So that long texts should be adjusted in a single

window

'''

# code has maximum lengths of 79 characters, can use backslash

# to break the line

list\_of\_subjects = [

'Physics', 'Chemistry', 'Mathematics', 'Biology', ‘Bio’, \

]

# Whitespaces, Trailing Commas, and String Quotes

I have tried to avoid extra white spaces, however there must be a single white space around both sides of an operator, one after the comma and none inside opening or closing of parenthesis. Both single quotes and double quotes are acceptable in [python web development](https://www.zenesys.com/python-development-company) and both has been used if quotes are needed inside quotes to avoid syntax error and extra backslash.

## For Example:-

# Examples of commas and whitespaces

x, y = 30, "text inside quotes"

z = 'text inside quotes'

if x == 30: print(x, y, z)

# how to use quotes inside quotes

text = "This text is using 'the single quote' inside double quote"

print(text)

# Naming Conventions

Use grammatically correct variable names, the class name starts with an uppercase and must follow camelCase convention If more than two words are to be used. In the same way, a function name has been joined with an underscore, and made lowercase. In method arguments, self is used as the first argument to declare an instance variable. If the function name clashes with a reserved argument, an underscore is used instead of a wrong spelling. Constants are declared in all capital letters.

## For Example: -

# class name follows camelcase convention

class StudentDetails:

def \_\_init\_\_(self, first\_name, last\_name):

self.first\_name = first\_name

self.last\_name = last\_name

# Method name, variable names in lowercase joined with an underscore

def grade(self, marks\_obtained):

# constants in capital

GRACE = 2

marks\_obtained = GRACE + marks\_obtained

if marks\_obtained > 90:

self.student\_grade = 'A'

elif marks\_obtained > 70:

student\_grade = 'B'

else:

student\_grade = 'C'

# Exception Handling for every critical situation

## For Example:-

try:

file = open('filename.txt')

file.write('Hello World')

except Exception as e:

print('Cannot open the file :', e)

finally:

# Make sure to close the file after

file.close()

# Documentation of a Method

Documenting every method with proper specification of parameters, return type, and data types. Multiple returns from a function are avoided, a single generic return is preferred.

## For Example:-

# documenting a function

def get\_grades(marks):

"""

Summary: getting grades from marks

Description: This function takes marks as an argument and returns grades

params:

marks(int) : marks obtained

:

grade(string) : grade achieved

"""

if marks > 90:

grade = 'A'

elif marks > 70:

grade = 'B'

else:

grade = 'C'

return grade