

## CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT USING JENKINS – RANDOM NUMBER QUEST



### A MINI DEVOPS PROJECT REPORT

by

## Team – A06

K.S.L.Sanjana(21JG1A0551) B.Rashmika(21JG1A0508) B.Prasanna(21JG1A0510) B.Pravallika(21JG5A0513)

Under the esteemed guidance of **Dr.P.V.S.Lakshmi Jagadamba Professor and HOD**

CSE Department

### Department of Computer Science and Engineering

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN**

[Approved by AICTE NEW DELHI, Affiliated to JNTUK Kakinada]

[Accredited by National Board of Accreditation (NBA) for B.Tech. CSE, ECE & IT – Valid from 2019-20 to 2021-22] Kommadi , Madhurawada, Visakhapatnam–530048

# 2023–2024

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



**CERTIFICATE**

This is to certify that the project report titled **“CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT USING JENKINS- RANDOM NUMBER QUEST”** is a

bonafide work of following III B.Tech. students in the Department of Computer Science a n d Engineering, Gayatri Vidya Parishad College of Engineering for Women affiliated to JNT University, Kakinada during the academic year 2023-2024 Semester-1.

**K.S.L.Sanjana**(21JG1A0551) **B.Rashmika** (21JG1A0508)

**B.Prasanna**(21JG1A0510) **B.Pravallika** (21JG5A0513)

### Dr.P.V.S.Lakshmi Jagadamba Professor and HOD

**CSE Department**

# TABLE OF CONTENTS

**TOPICS PAGENO.**

|  |  |  |
| --- | --- | --- |
|  | **Abstract** |  |
| **1.** | **WEB APPLICATION** |  |
|  | 1.1 Introduction of Web Application | 1 |
| 1.2 Hardware and Software requirements | 1-3 |
| 1.3 Methodology | 3-4 |
| 1.4 Model architecture diagram | 5-6 |
| 1.5 Implementation | 7-11 |
| 1.6 Results | 12-14 |
| **2.** | **CICD USING JENKINS** |  |
|  | 2.1 Git Bash | 15-18 |
|  | 2.2 Jenkins |  |
|  | 2.2.1 Installation of Jenkins | 18-21 |
| 2.2.2 Unlocking Jenkins | 21-23 |
|  | 2.2.3 Jenkins Plugins | 23-24 |
| 2.2.4 Chrome extensions | 24-26 |
| 2.2.5 Building a job in Jenkins | 26-28 |
| 2.2.6 Implementation of Jenkins Pipelines | 28-34 |
| 2.2.7 Results | 34-38 |
| **3.** | **CONCLUSION** | **39-40** |

# ABSTRACT

A simple web application called "Random Number Quest," where the user is tasked with guessing a randomly generated number. The player is given up to 10 attempts to guess the secret number, and the application provides feedback after each guess, including whether the guess is too low, too high, or if the player has won or lost the game. The code for this project is managed in a Git repository, and Jenkins is set up to automate the build process and provide feedback on the code quality and correctness. This report provides the detailed steps to deploy our project along with their respective source codes.

### Key words:

HTML, CSS, JavaScript, GitHub, GitBash, Team Repository, Git push commands, Jenkins, Jenkins Plugins, Build, Deployment.

# WEB APPLICATION

## Introduction of Web Application:

A web application called "Random Number Quest". This is a user-friendly interface which waits for the input from user and compares with the randomly generated number. After comparing , if the user guess it correctly it sets user status as winner and displays. If not it will compare and displays the number the number is greater or lower than the generated number and displays the number of chances we had to perfrom and our applications offers 10 chances to guess the number correctly. We enabled the replay button to start the game again as per users wish. Codes of this web application are maintained in git hub repository.

The project is a basic web-based guessing game that demonstrates how to create a simple interactive web application using HTML, CSS, and JavaScript. It is a stress free game. It's a fun and educational game where the user's goal is to guess the correct random number within a limited number of attempts and the coding of this web application is maintained in git and managed into Jenkins by integrating with git hub repository url.

* **Game Concept**:

The project centers around a number guessing game where players attempt to guess a secret random number between 1 and 100. They have a maximum of 10 attempts to guess the correct number.

* **Automation and Feedback**:

Jenkins provides automated feedback on the code quality and correctness. It reports build status, errors, and test results.

By automating the build process, the project ensures that code changes are validated and that the application remains functional.

* **Collaborative Development**:

The use of Git and Jenkins promotes collaboration among developers by enabling multiple contributors to work on the project simultaneously and ensuring that changes do not break the application

## Hardware and Software requirements:

### Hardware requirements:

* **Processor:** A modern, multi-core processor capable of running the application efficiently. At minimum, a dual-core processor is recommended.
* **Memory (RAM):** The amount of memory needed will depend on the number of concurrent users and the amount of data being processed. A minimum of 4 GB of RAM is recommended

for a small-scale system, but larger systems may reqsuire more memory

* **Network Interface:** The system should be connected to a stable and reliable network, such as a local area network (LAN) or a wide area network (WAN).
* **Display:** A high-resolution display is recommended to ensure that the user interface is clear and easy to read.
* **Input Devices:** A keyboard and mouse are required for inputting data into the system.
* **Mobile Devices:** If the system is designed to be accessed from mobile devices, it should be compatible with different types of mobile devices and operating systems. It should also be optimized for mobile devices, with a responsive and mobile-friendly user interface.
* Overall, the hardware requirements will depend on the scale of the system, the number of concurrent users, and the amount of data being processed. It is recommended to consult with

a hardware specialist to determine the optimal hardware configuration for the system.

### Software requirements:

* **Text Editors:**
  + - 1. **Notepad (Windows):** A basic text editor that comes pre-installed on most Windows computers.
      2. **Visual Studio Code:** A powerful and popular source code editor with HTML syntax highlighting, extensions, and a lot of features.
      3. **Sublime Text:** A highly customizable text editor with support for many programming languages, including HTML.
* **Online HTML Editors:**

There are various online HTML editors that allow you to create, edit, and preview HTML documents directly in your web browser. Some examples include CodePen, JSFiddle, and HTML Online.

* **HTML**: Provides the structure of the web page.
* **CSS**: Defines the styling and appearance of the web page elements.
* **JavaScript**: Implements the game logic, manages user input, and displays feedback to the player.
* **Version Control with Git**:

The project code is stored and managed in a Git repository hosted on a Git platform (e.g., GitHub, GitLab, Bitbucket). This allows for version control, collaborative development, and easy tracking of code changes.

* **Continuous Integration with Jenkins:**

Jenkins, a popular CI/CD tool, is integrated into the project to automate the build and testing process.

A Jenkins job is set up to retrieve the code from the Git repository and execute a build pipeline defined in a Jenkinsfile.

The build pipeline typically involves tasks such as code compilation, running tests, and archiving artifacts.

## Methodology:

* **Set Up Your Development Environment:**

Choose a programming language for your game (e.g., Python, JavaScript, Java).

Install the necessary development tools and libraries for your chosen language.

Create a local development environment for coding and testing.

* **Develop the Random Number Guessing Game:**

Write the code for the random number guessing game.

Test the game locally to ensure it works as expected.

* **Version Control:**

Set up a version control system (e.g., Git).

Create a repository for your game and commit your code.

* **Jenkins Installation and Configuration:**

Install Jenkins on a server or in a cloud environment.

Access the Jenkins web interface and set it up according to your environment.

* **Jenkins Plugins Installation:**

Install any necessary plugins, such as Git, GitHub, and build tool-specific plugins, via the Jenkins Plugin Manager.

* **Create a Jenkins Job:**

Create a new Jenkins job to automate the build and deployment process.

Choose a job type, such as a Freestyle project.

* **Configure Source Code Management:**

Configure your Jenkins job to fetch the game's source code from your version control repository (e.g., Git).

Specify the branch you want to build from.

* **Set Up the Build Step:**

Add a build step to your Jenkins job that compiles or runs your game.

Use a shell script or batch command to execute the game.

* **Configure Post-Build Actions:**

Set up post-build actions based on your deployment requirements. This might involve packaging, containerizing, or copying the game to the deployment environment.

Configure the Jenkins job to notify you or relevant stakeholders about the build and deployment status.

* **Deployment Pipeline (Continuous Deployment):**

Set up a deployment pipeline within Jenkins that takes your code from a source code repository and deploys it to the target environment.

Utilize Jenkins plugins or custom scripts for deployment tasks.

* **Test the Deployment:**

Ensure that the game is deployed successfully in the target environment.

Test its functionality and accessibility in the deployment environment.

* **Automate the Process:**

Configure Jenkins to automatically trigger builds and deployments whenever changes are pushed to the source code repository.

This can be achieved through webhooks, periodic builds, or other automation triggers.

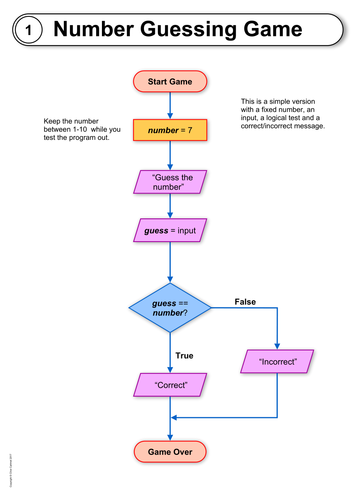
* **Monitoring and Continuous Improvement:**

Set up monitoring to keep an eye on your Jenkins job and the deployed game.

Continuously monitor and improve the game, the deployment process, and the Jenkins job as needed.

By following this methodology, we can be able to develop a random number guessing game, integrate it with Jenkins, and automate the build and deployment process, allowing you to deliver your game to users more efficiently and consistently.

## Model architecture diagram:



### Fig 1.4. Model Architecture diagram for Random Number Quest

* **User Interface:**

The game starts with a user interface that displays information to the player and collects their inputs.

* **Game Logic:**

The core of the game is the game logic. It includes the following components:

* **Random Number Generator:** This generates a random number within a specified range.
* **Player Input Handling:** This component receives the player's guesses and other inputs.
* **Feedback Generator**: It compares the player's guess with the randomly generated number and provides feedback (e.g., too high, too low, correct).
* **Game State Management:** It tracks the game's progress, such as the number of attempts and whether the game is won or lost.
* **User Feedback:**

After each guess, the game provides feedback to the player. This is usually displayed in the user interface.

* **Win or Lose Condition:**

The game logic checks whether the player's guess is correct. If the guess is correct, the player wins. If not, the player continues guessing until they either win or reach the maximum allowed attempts (lose condition).

* **Game Over:**

Once the game is over (either the player wins or loses), the user interface displays the final result and asks if the player wants to play again.

## Implementation:

### Source code for index.html:

<!DOCTYPE html>

<!-- Coding by CodingLab || www.codinglabweb.com -->

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Game in HTML CSS & JavaScript</title>

<link rel="stylesheet" href="style.css" />

</head>

<body>

<div class="wrapper">

<header>Guess a number from 1 to 100</header>

<p class="guess"></p>

<div class="input-field">

<input type="number" />

<button>Check</button>

</div>

<p>You have <span class="chances">10</span> chances</p>

</div>

<script src="script.js" defer></script>

</body>

</html>

**15.2 Source code for css:**

/\* Import Google font - Poppins \*/

@import url("https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;500;600;700&display=swap");

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

font-family: "Poppins", sans-serif;

}

body {

min-height: 100vh;

display: flex;

align-items: center;

justify-content: center;

background: #4a98f7;

}

.wrapper {

padding: 30px 40px;

border-radius: 12px;

background: #fff;

text-align: center;

box-shadow: 0 5px 10px rgba(0, 0, 0, 0.1);

}

.wrapper header {

font-size: 18px;

font-weight: 400;

color: #333;

}

.wrapper p {

color: #333;

}

.wrapper .input-field {

display: flex;

justify-content: center;

gap: 20px;

margin: 25px 0;

}

.input-field input,

.input-field button {

height: 50px;

width: calc(100% / 2 - 20px);

outline: none;

padding: 0 20px;

border-radius: 8px;

font-size: 18px;

}

.input-field input {

text-align: center;

color: #707070;

width: 110px;

border: 1px solid #aaa;

}

input::-webkit-inner-spin-button,

input::-webkit-outer-spin-button {

display: none;

}

.input-field input:disabled {

cursor: not-allowed;

}

.input-field button {

border: none;

background: #4a98f7;

color: #fff;

cursor: pointer;

transition: 0.3s;

}

.input-field button:active {

transform: scale(0.97);

}

}

.wrapper .input-field {

display: flex;

justify-content: center;

gap: 20px;

margin: 25px 0;

}

.input-field input,

.input-field button {

height: 50px;

width: calc(100% / 2 - 20px);

outline: none;

padding: 0 20px;

border-radius: 8px;

font-size: 18px;

}

.input-field input {

text-align: center;

color: #707070;

width: 110px;

border: 1px solid #aaa;

}

input::-webkit-inner-spin-button,

input::-webkit-outer-spin-button {

display: none;

}

.input-field input:disabled {

cursor: not-allowed;

}

.input-field button {

border: none;

background: #4a98f7;

color: #fff;

transition: 0.3s;

}

.input-field button:active {

transform: scale(0.97);

}

### 

cursor: pointer;

transition: 0.3s;

}

.input-field button:active {

transform: scale(0.97);

}

**15.3 Source code for javascript:**

// Get the DOM elements and initialize the game

const input = document.querySelector("input"),

guess = document.querySelector(".guess"),

checkButton = document.querySelector("button"),

remainChances = document.querySelector(".chances");

// Set the focus on input field

input.focus();

let randomNum = Math.floor(Math.random() \* 100);

chance = 10;

// Listen for the click event on the check button

checkButton.addEventListener("click", () => {

// Decrement the chance variable on every click

chance--;

// Get the value from the input field

let inputValue = input.value;

// Check if the input value is equal to the random number

if (inputValue == randomNum) {

// Update guessed number, disable input, check button text and color.

[guess.textContent, input.disabled] = ["Congratulations", true];

[checkButton.textContent, guess.style.color] = ["Replay", "#333"];

//Check if input value is > random number and within 1-99 range.

} else if (inputValue > randomNum && inputValue < 100) {

// Update the guess text and remaining chances

[guess.textContent, remainChances.textContent] = ["Your guess is high", chance];

guess.style.color = "#333";

//Check if input value is < random number and within 1-99 range.

} else if (inputValue < randomNum && inputValue > 0) {

// Update the guessed number text and remaining chances

[guess.textContent, remainChances.textContent] = ["Your guess is low", chance];

guess.style.color = "#333";

// If the input value is not within the range of 1 to 99

} else {

// Update the guessed number text, color and remaining chances

[guess.textContent, remainChances.textContent] = ["Your number is invalid", chance];

guess.style.color = "#DE0611";

}

// Check if the chance is zero

if (chance == 0) {

//Update check button, disable input, and clear input value.

// Update guessed number text and color to indicate user loss.

[checkButton.textContent, input.disabled, inputValue] = ["Replay", true, ""];

[guess.textContent, guess.style.color] = ["You lost the game", "#DE0611"];

}

if (chance < 0) {

window.location.reload();

}

});

// Update guessed number, disable input, check button text and color.

[guess.textContent, input.disabled] = ["Congratulations", true];

[checkButton.textContent, guess.style.color] = ["Replay", "#333"];

//Check if input value is > random number and within 1-99 range.

} else if (inputValue > randomNum && inputValue < 100) {

// Update the guess text and remaining chances

[guess.textContent, remainChances.textContent] = ["Your guess is high", chance];

guess.style.color = "#333";

//Check if input value is < random number and within 1-99 range.

} else if (inputValue < randomNum && inputValue > 0) {

// Update the guessed number text and remaining chances

[guess.textContent, remainChances.textContent] = ["Your guess is low", chance];

guess.style.color = "#333";

// If the input value is not within the range of 1 to 99

} else {

// Update the guessed number text, color and remaining chances

[guess.textContent, remainChances.textContent] = ["Your number is invalid", chance];

guess.style.color = "#DE0611";

}

// Check if the chance is zero

if (chance == 0) {

//Update check button, disable input, and clear input value.

// Update guessed number text and color to indicate user loss.

[checkButton.textContent, input.disabled, inputValue] = ["Replay", true, ""];

[guess.textContent, guess.style.color] = ["You lost the game", "#DE0611"];

}

if (chance < 0) {

window.location.reload();

}

});

|  |
| --- |
|  |
|  |  |

## Results:

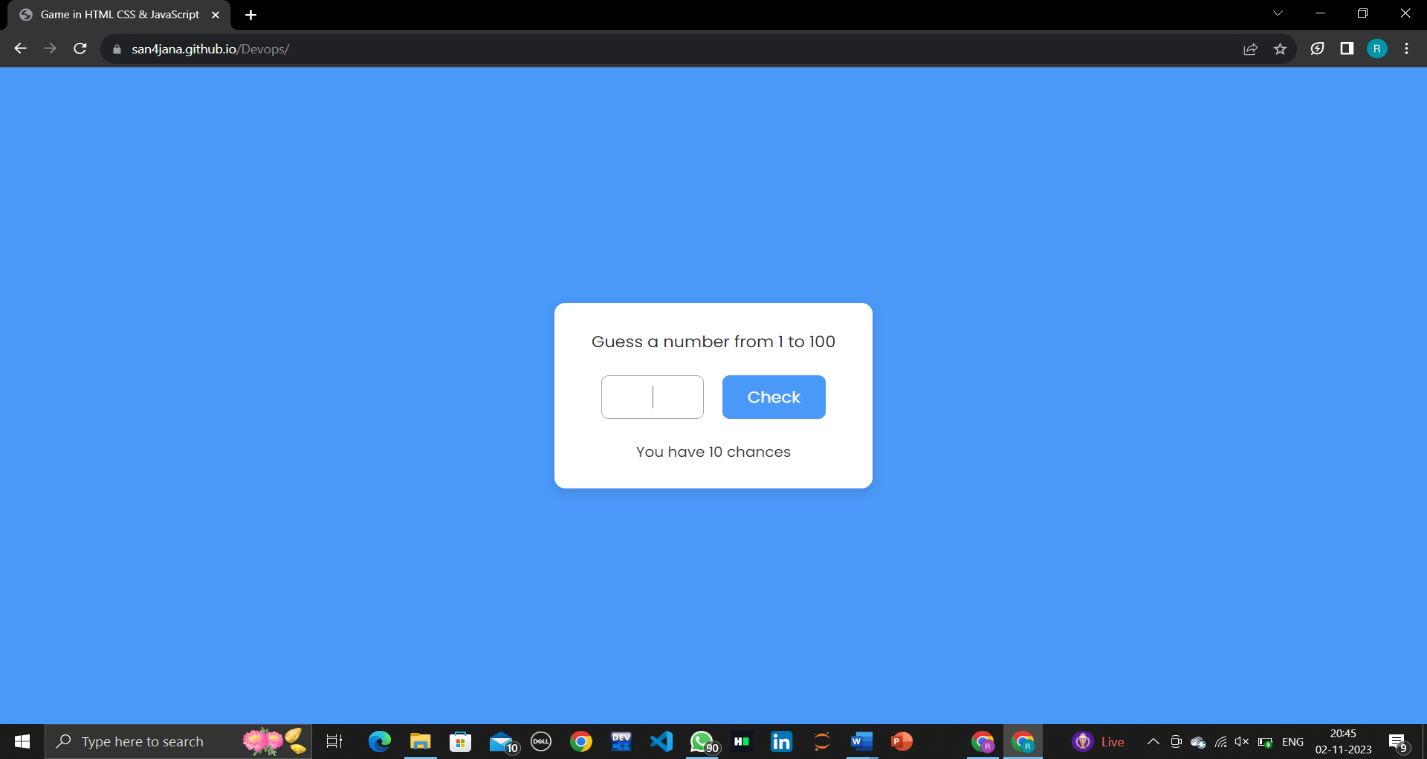
## Results (output description):

## 

The game's user interface consists of a title, an input field where the user enters their guess, a "Submit" button to submit the guess, and a message area to display feedback. The user is required to guess a random number between 1 and 100.

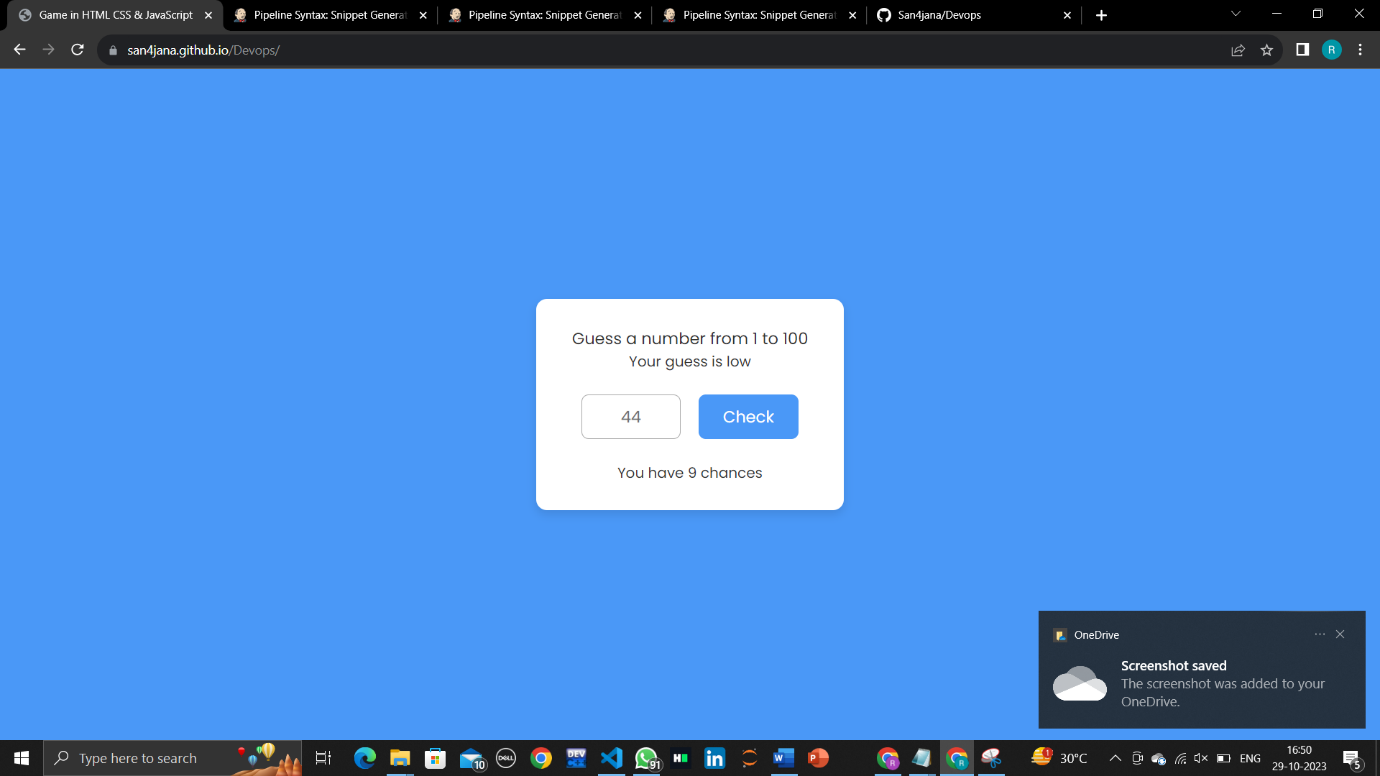
* If the user's guess matches the secret number, a congratulatory message is displayed, indicating the number of attempts it took to guess correctly.
* If the user exceeds 10 attempts without guessing correctly, a message reveals displaying that you lost the game and shows you the replay button.
* If the user's guess is too low, a message suggests trying a higher number.
* If the user's guess is too high, a message suggests trying a lower number.

### Home page:

****

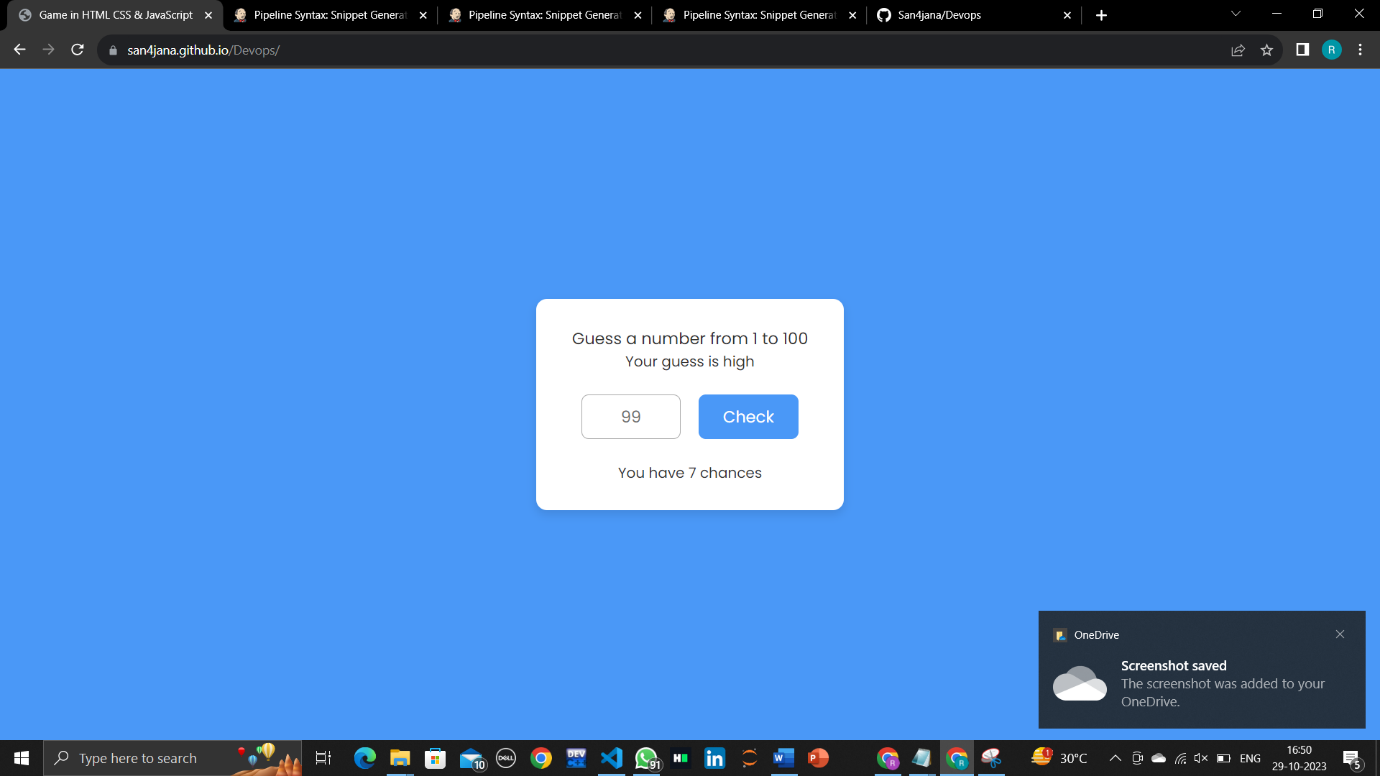
**Fig.1.6.1 Implementing Random number quest web using Browser**

### When your guess is low:



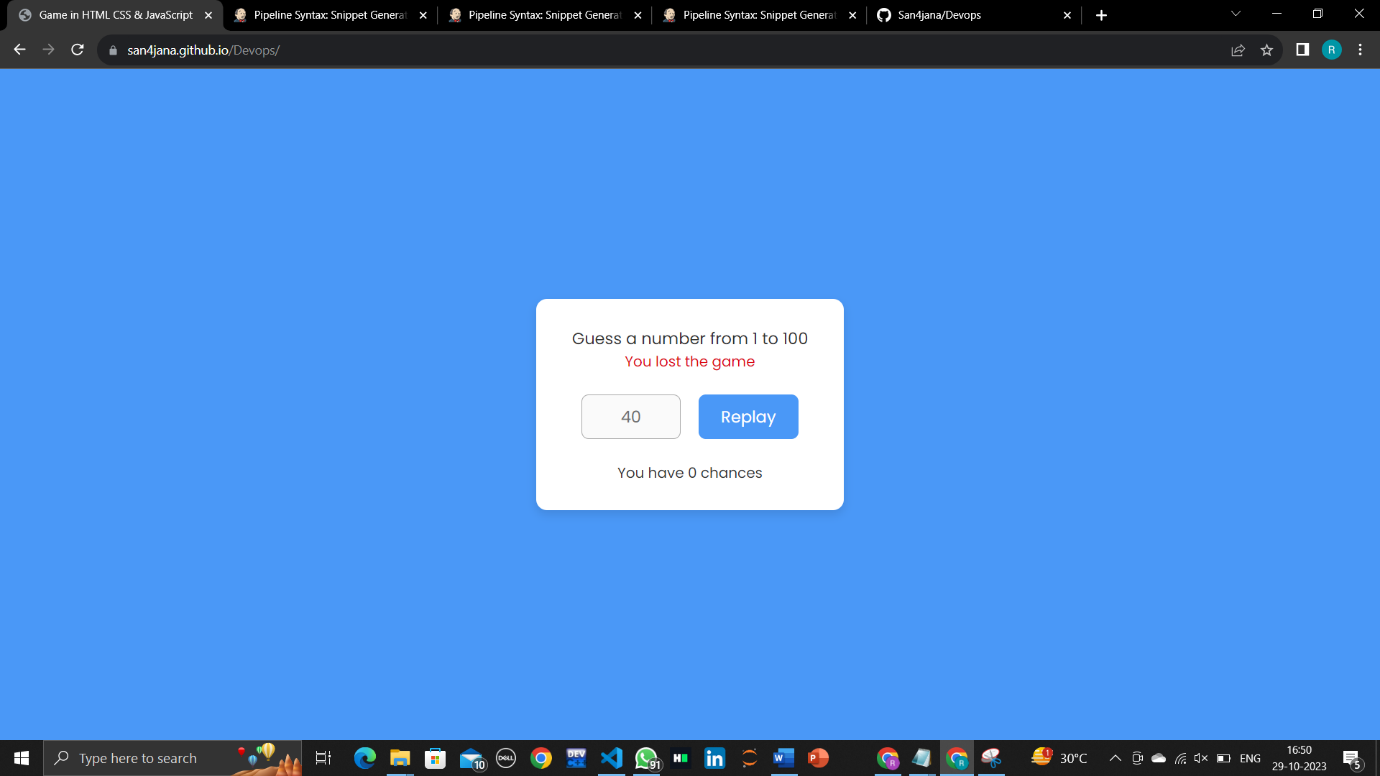
**Fig.1.6.2 showing when the guessed number is low**

* + 1. **When your guess is high:**



**Fig.1.6.3 showing when the guessed number is high**

### When you lost the game:



**Fig.1.6.4 shows when you lost the game**

# CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT USING JENKINS

## Git Bash:

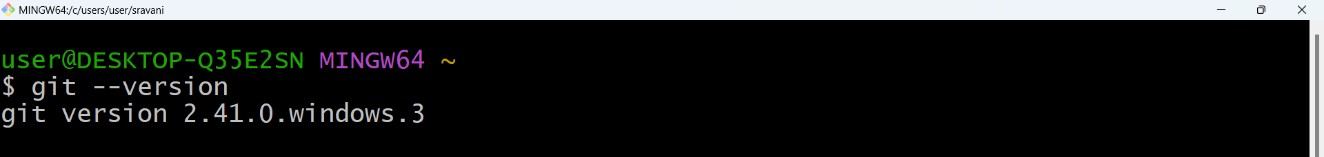
Git Bash is an application for Microsoft Windows environments which provides an emulation layer for a Git command line experience. Bash is an acronym for Bourne Again Shell. A shell is a terminal application used to interface with an operating system through written commands.

## Installation and Setup of Git:

* + - 1. Download GIT from the website below based on your system’s operating system and configuration.



* + - 1. Install GIT by clicking next for a few times.
      2. After installation is complete go to start menu and search for Git Bash and open it.
      3. Check the Git version using the command git –version.



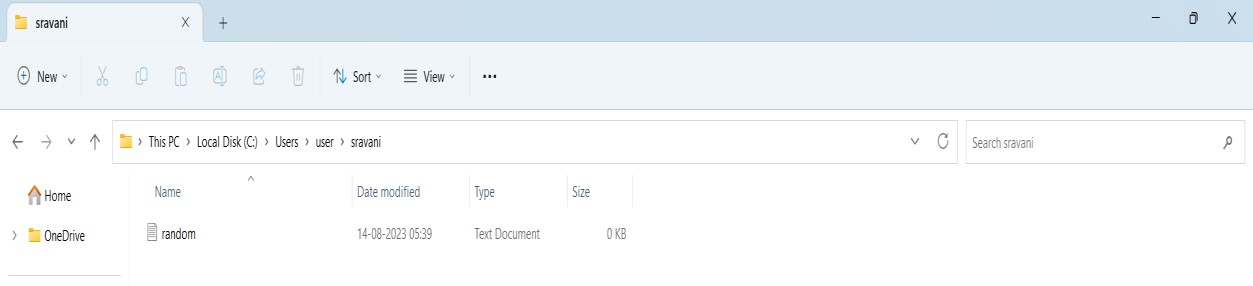
* + - 1. Create a local directory using the following commands:

### mkdir directory\_name cd directory\_name

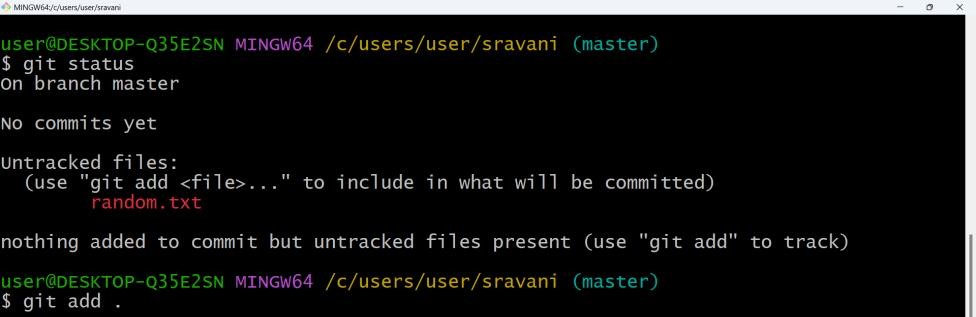
* + - 1. The next step is to initialize the directory using the git init command as shown below.



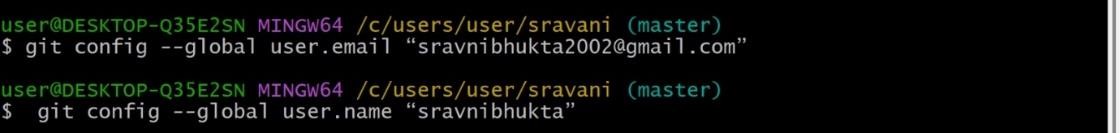
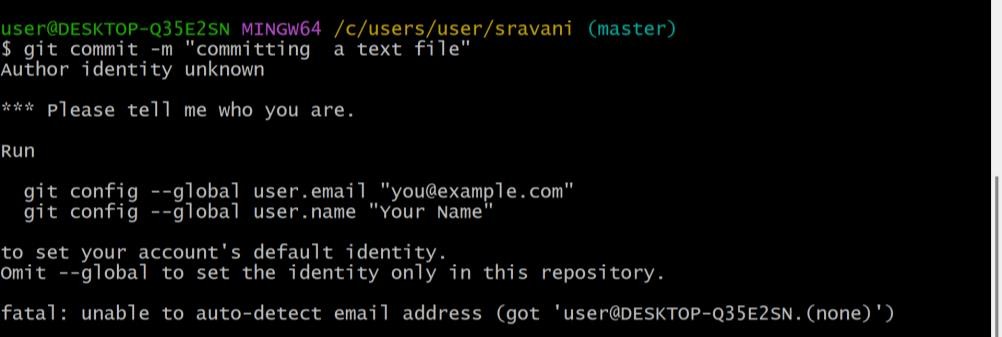
* + - 1. Create a text file inside the newly created local directory, write some content and save it.



* + - 1. Enter the Git bash interface and type in the following command to check the status of the files inside the local directory using: **git status**
      2. **git add [file name]** or **git add .** tracks the respective files and are ready to be committed into the GitHub repository.
      3. Every time you make modifications to a particular file, you need to add it using git add command to make it ready to be committed.



* + - 1. Next, make a commit using **git commit -m "committing message"** as shown below.
      2. Link the Git to a GitHub Account using following commands as shown: **git config --global user.email “your email linked to GitHub account” git config --global user.name “your GitHub username”**



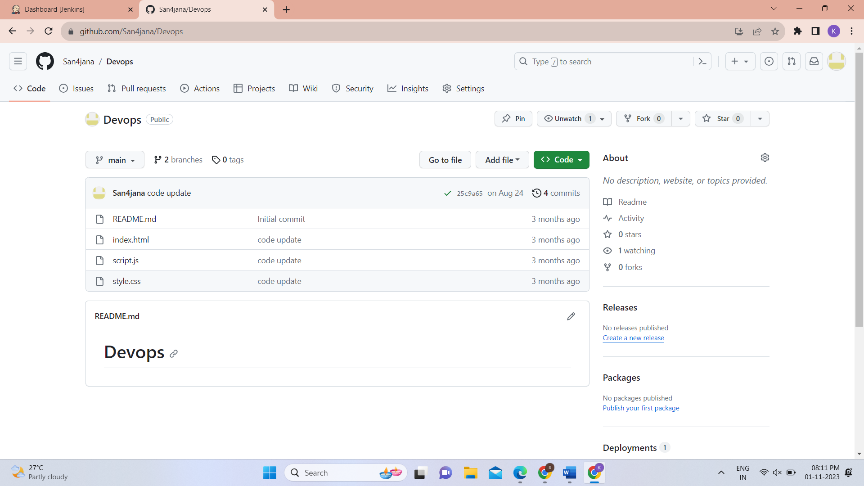
* + - 1. Open your GitHub account and create a **new repository** with any name of your choice.
      2. **Public repositories** are accessible to everyone on the internet while the **Private repositories**

are only accessible to you and the people you explicitly share access to.

* + - 1. Push the project folder using following commands.
         1. **git clone https://github.com/San4jana/Devops.git**
         2. **git status**

### git add .

* + - * 1. **git commit -m "code update"**
        2. **git push origin main**



## Jenkins:

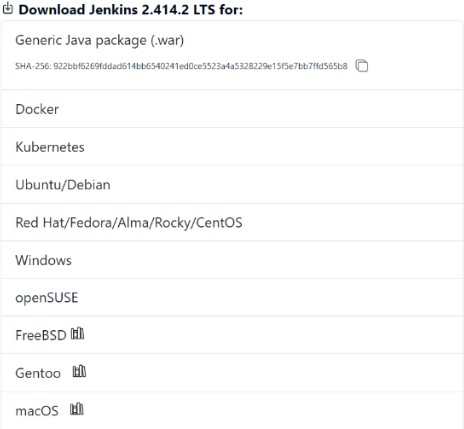
There are 2 steps involved in setting up Jenkins completely. They are,

* Download and installation of Jenkins software
* Unlocking Jenkins with localhost

## Steps to download and install Jenkins:

* + - 1. Go to the website [**https://www.jenkins.io/download/**](https://www.jenkins.io/download/) where we can get Jenkins software for any type of Operating system (Windows, Linux, Mac, etc..). Select windows for Jenkins

2.414.2 LTS.

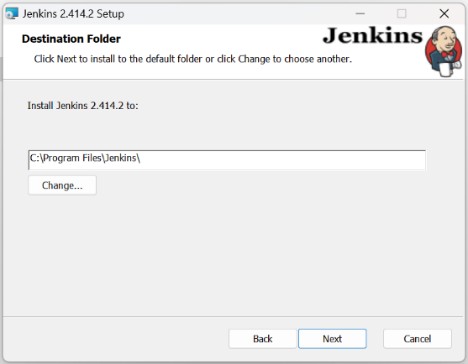


* + - 1. After selecting windows, Jenkins.exe file starts downloading.

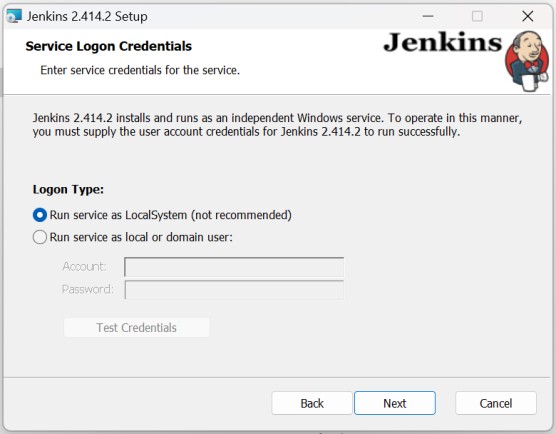
Go to Downloads -> Double click on Jenkins.exe file. Click on Next.



* + - 1. Now you will be asked to set the destination for the folder. There is no need to change the path. Click on Next.



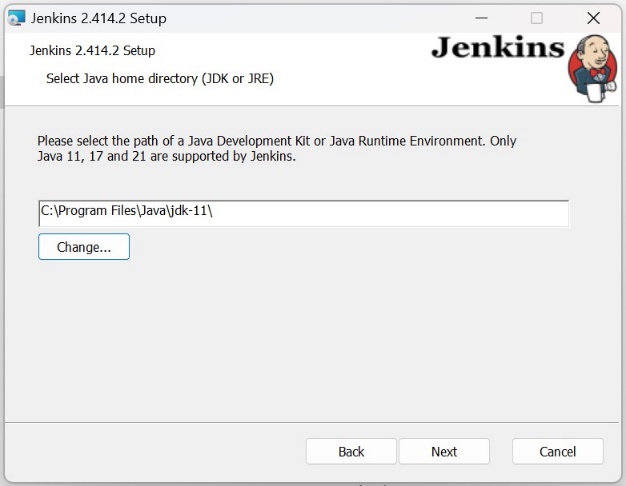
* + - 1. Now you will be asked to enter the services logon credentials. Select Run service as LocalSystem and click on Next.



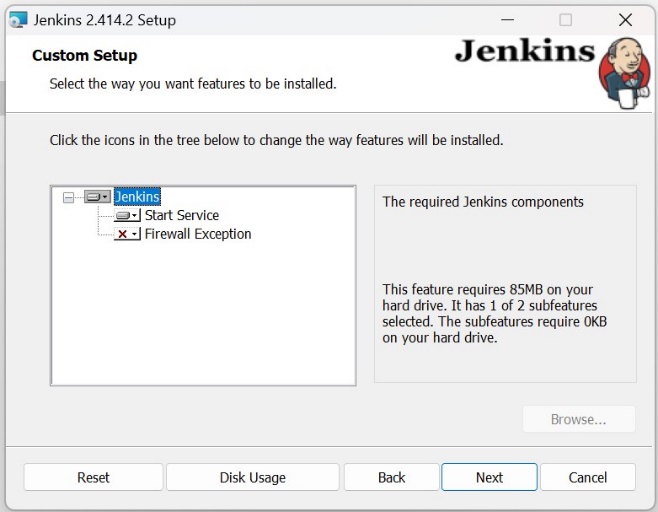
* + - 1. Now you will be asked to select a port number for localhost. By default it is 8080 if it is not available, then you have to choose one from the available ports in your device. You can use that number with localhost when the test post result is success. Click on Next.



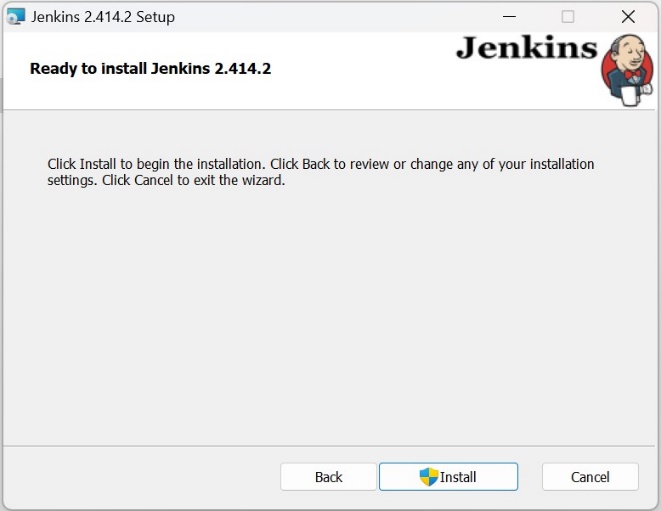
* + - 1. As Jenkins runs on Java Environment, we must give Java path to Jenkins. Jenkins supports only Java 11, 17 or 21. After giving path of Java, Click on Next.



* + - 1. Now a custom setup window opens. Click on Next without making any changes in it.



* + - 1. Click on Install. Now the installation of Jenkins begins.

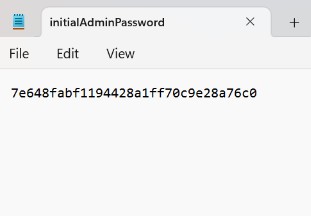
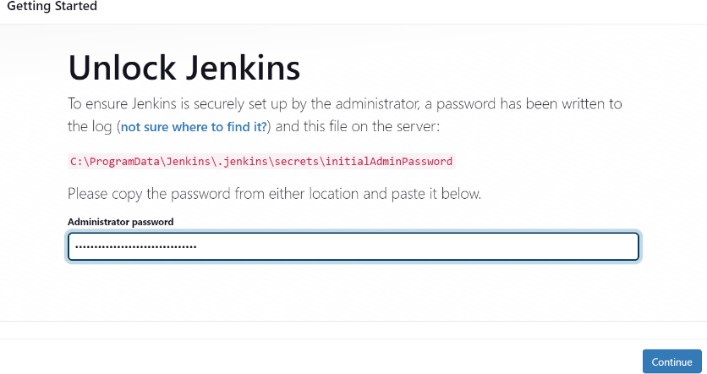


* + - 1. After the completion of Installation, Click on Finish.

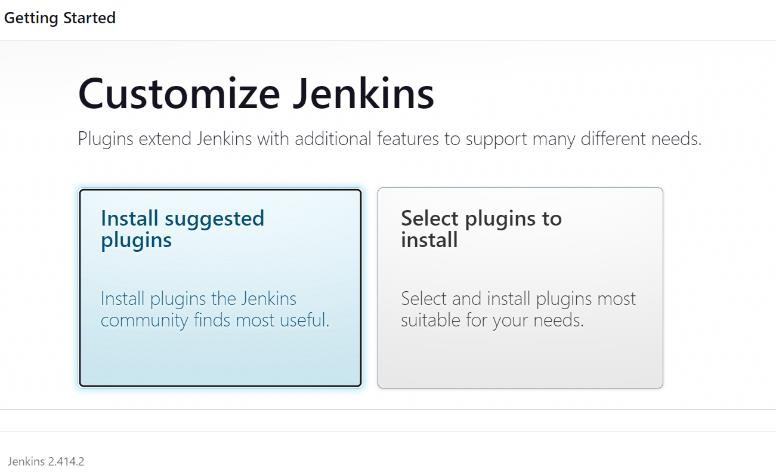


## Steps to unlock the Jenkins after Installation:

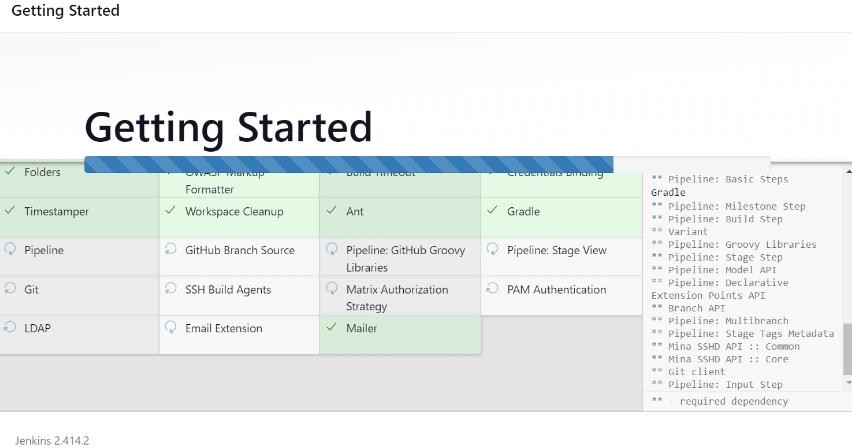
* + - 1. Open your web browser. Type localhost:8080 (your port number). After pressing Enter you will be opened with a Getting started window asking for a password. To find the password open the location mentioned and copy paste the code. Click on Continue.



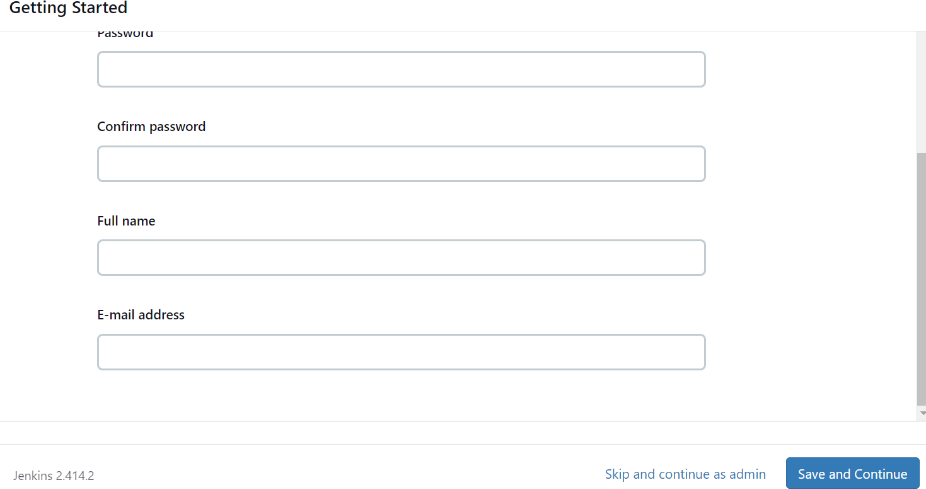
* + - 1. Select Install suggested plugins.



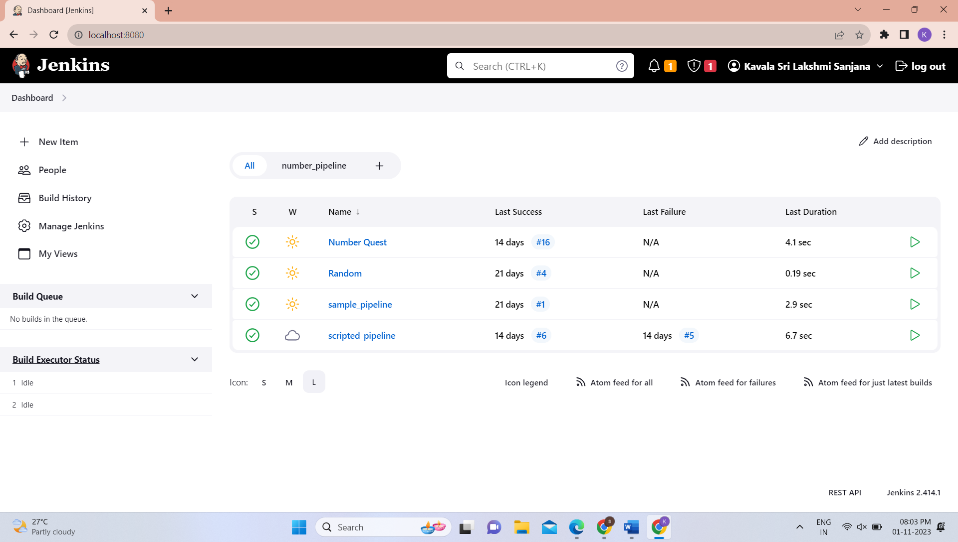
* + - 1. Now all the required plugins will be downloaded one by one.

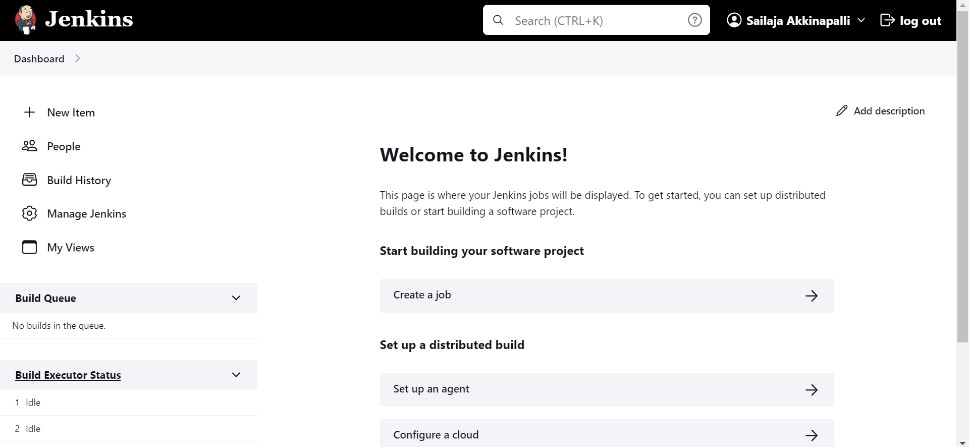


* + - 1. Now create an account by filling all the required details. Click on Save and Continue after filling all the details.



* + - 1. After clicking on Save and Continue, you successfully completed setting up your profile in Jenkins.

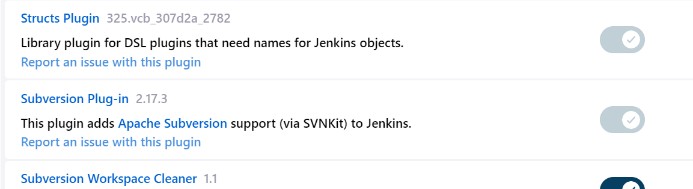
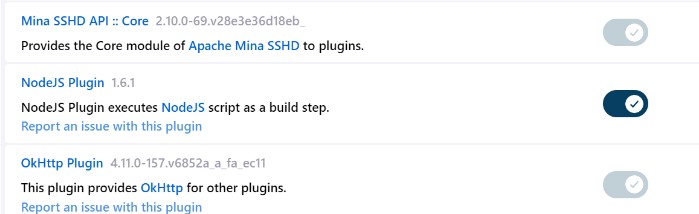
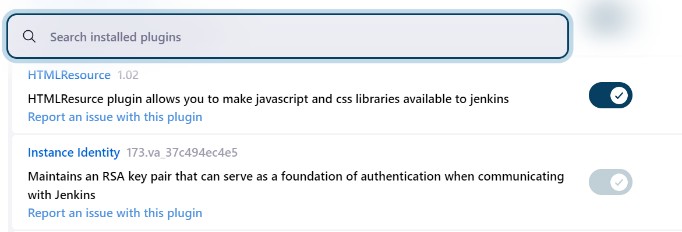
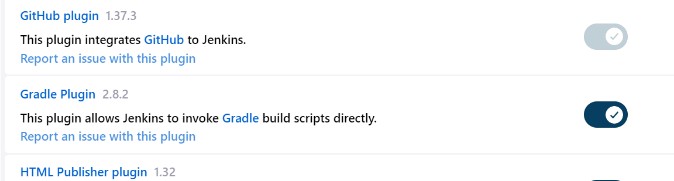




## Installation of additional required plugins in Jenkins:

To make Jenkins deploy your project correctly, you should install some more additional plugins. Here is the list of plugins we installed to deploy our project and view the output.

* Docker plugin
* HTML Publisher plugin
* NodeJS Plugin
* Subversion Workspace Cleaner
* HTMLResource
* Post build task



## Chrome Extensions required:

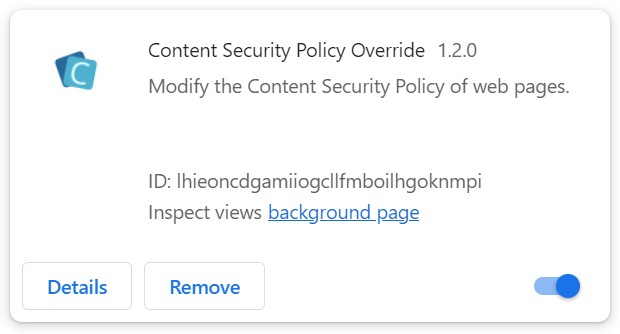
Sometimes when we execute our web page in Jenkins, Content Security Policy (CSP) blocks content in web pages as part of its core security function. CSP is designed to enhance the security of web applications by preventing various types of attacks, including cross-site scripting (XSS) attacks, data injection, and other code execution vulnerabilities.

### How to stop content blocking of a Web page in Jenkins ?

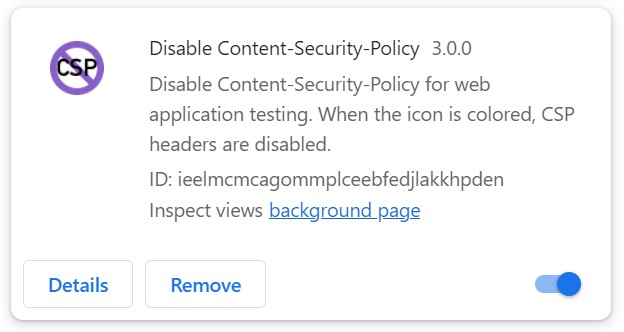
In order to stop content blocking in our Web page, using Chrome extensions to disable Content Security Policy (CSP) can be a useful tool during web development or while accessing certain websites, but it should be used with caution, and primarily for specific testing or debugging purposes.

### What are the required Chrome extensions to disable Content Security Policy(CSP) ?

* + - * 1. **Content Security Policy Override:**

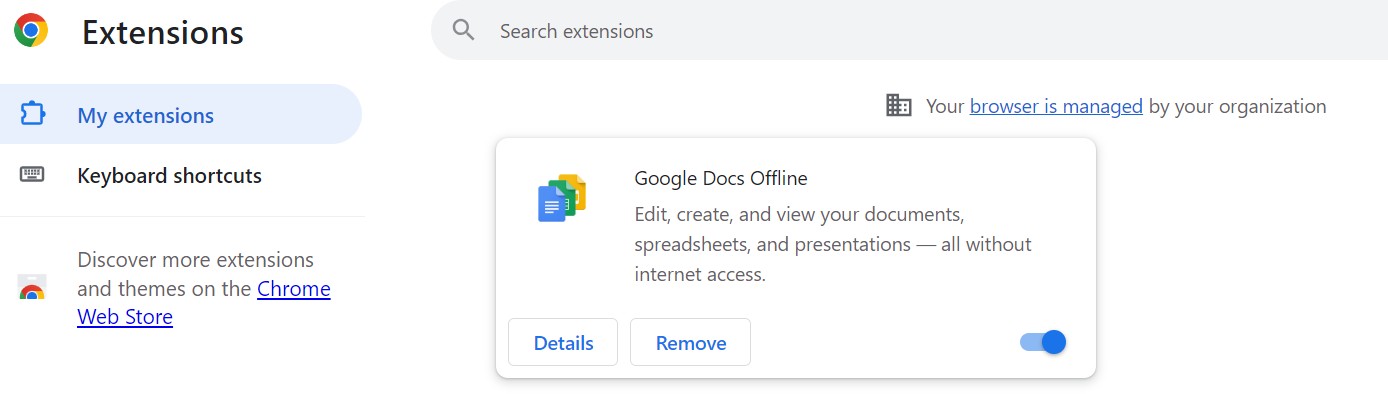
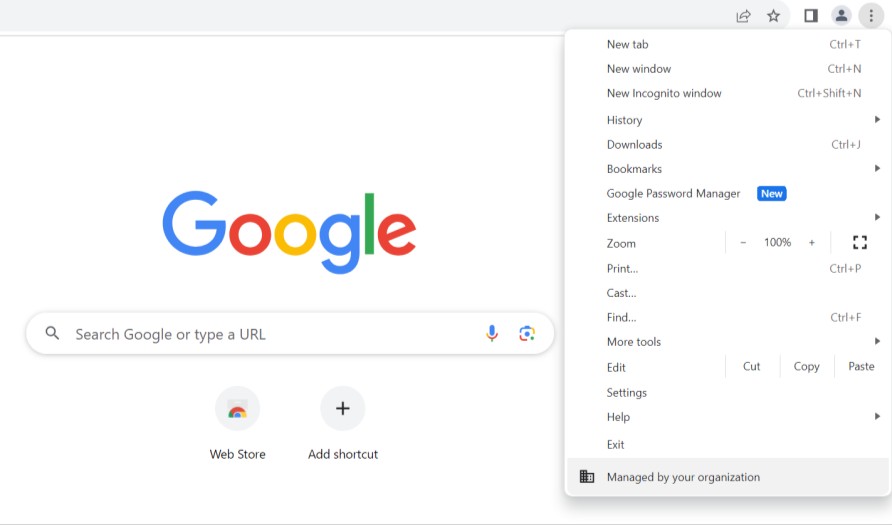


### Disable Content-Security-Policy:

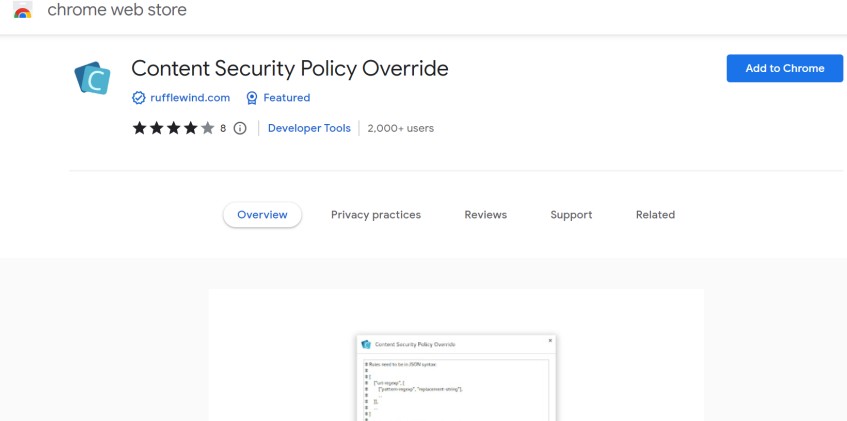


* + - 1. **How to download the required Chrome extensions ?**
* Open your Google Chrome browser. Click on 3 dots at the right upper corner.

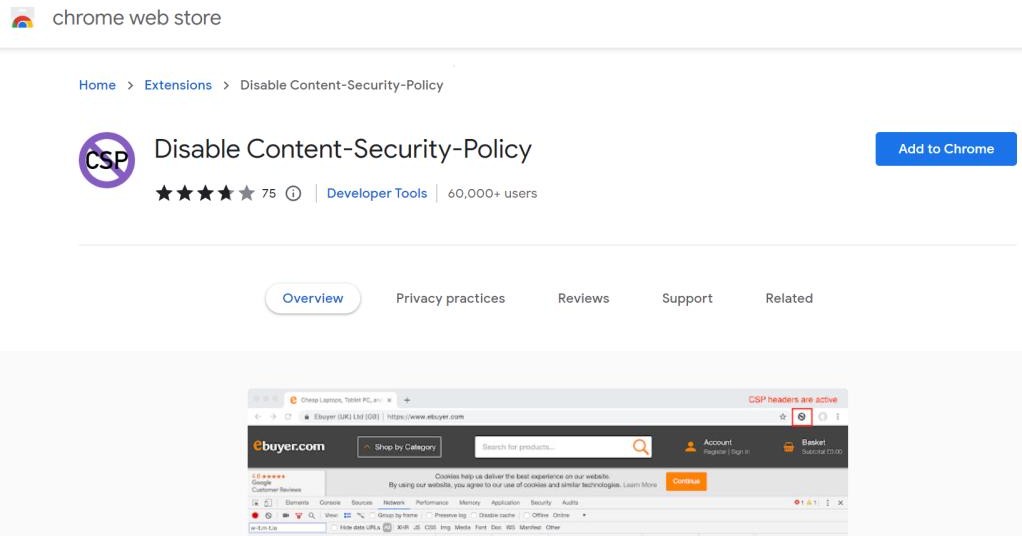
Select Extensions - > Manage Extensions. It shows the extensions which already exists in your Chrome page.



* Click on Chrome Web Store. Type **Content Security Policy Override** in the search bar provided at left side. Select Extension in the filter below the search bar.

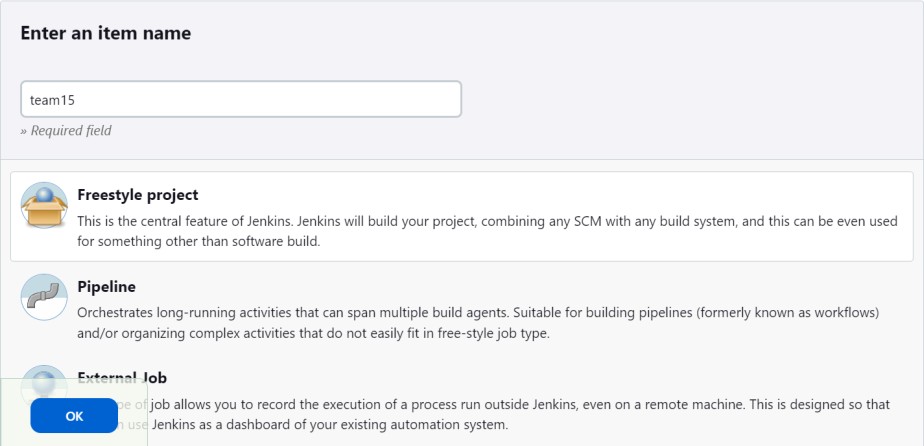


* Click on Add to Chrome. Click on Add Extension when pop-up window appears.
* After downloading the extension, Go to My extensions. You will find the downloaded extension enabled.
* In the same way download another required extension **Disable Content-Security-Policy**.

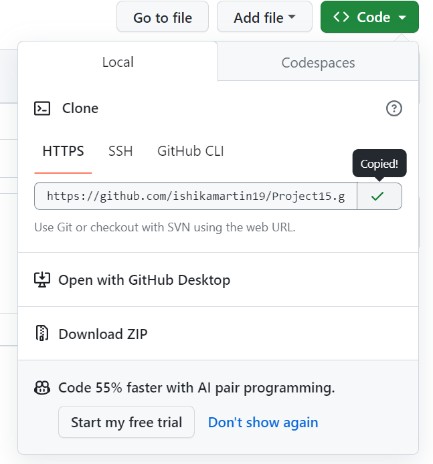
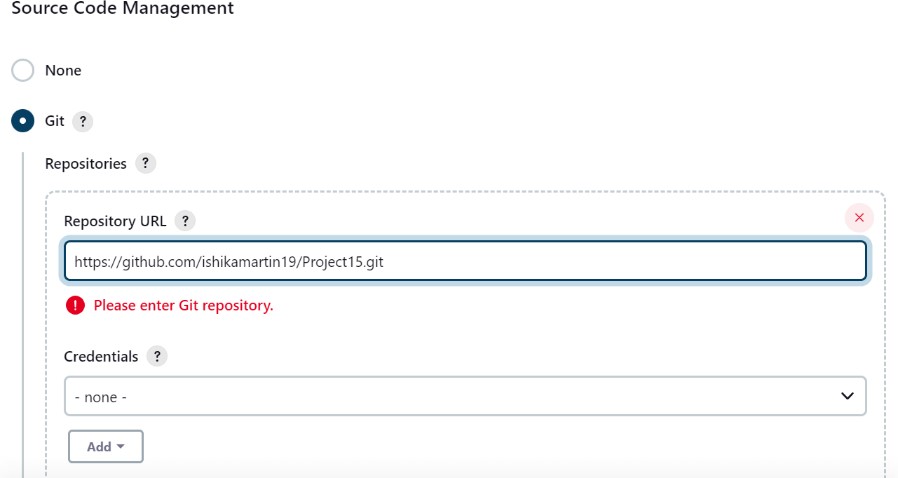


## Building a job in Jenkins:

* Open Jenkins Dashboard. Click on New Item.
* Enter a name for Job. Select Freestyle project and Click OK.



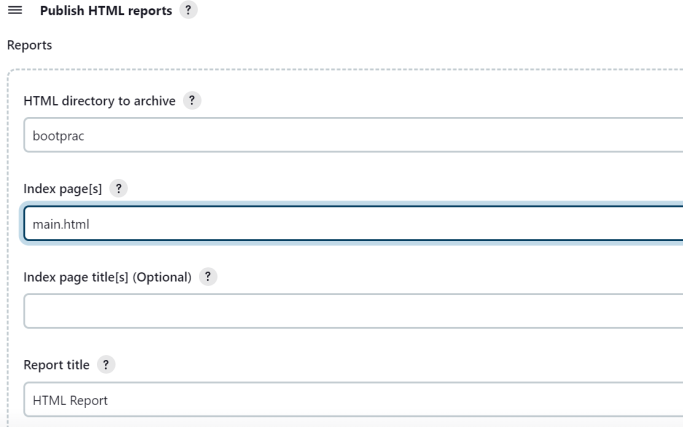
* Scroll down until you reach Source Code Management. Select Git.
* Now go to your GitHub account and copy the URL of your Repository in which your project is present.
* Paste the copied URL in Repository URL section.



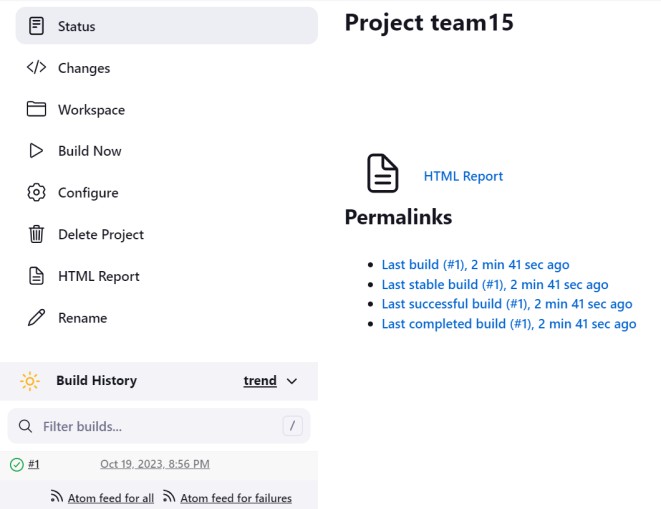
* Scroll down, change the branch into your branch name either it is main or others.



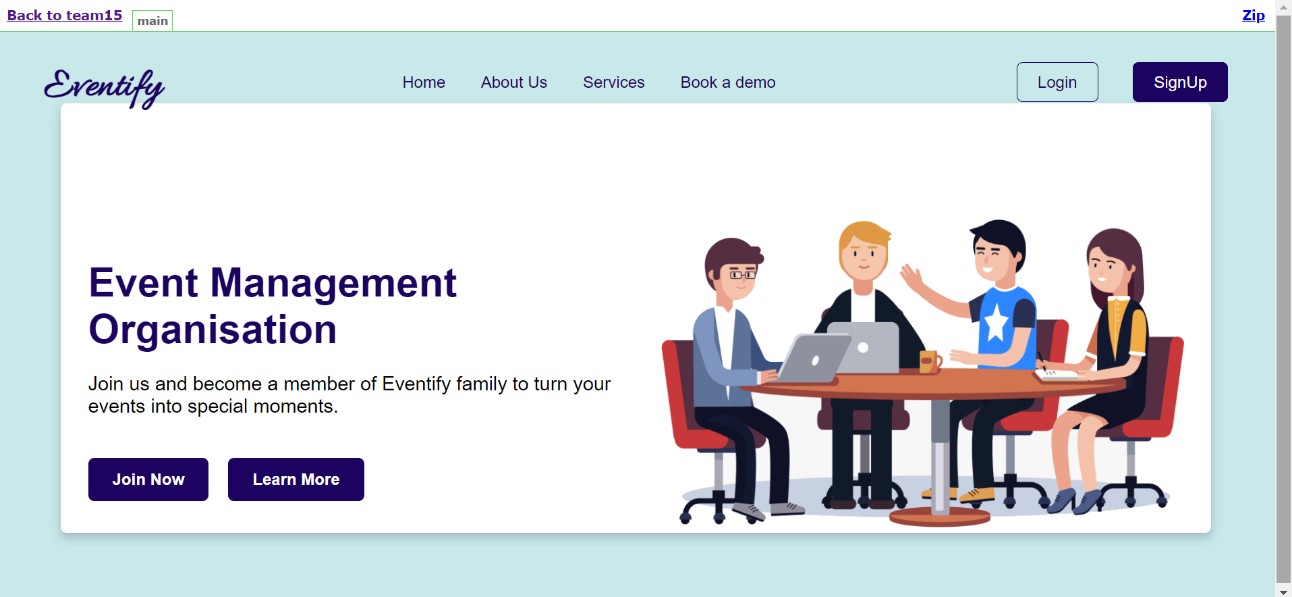
* Again scroll down, Post-build Actions - > Add post-build action - > Publish HTML Reports. Click on Add.
* Give your project folder name as HTML directory and give name of your first html page as Index page.
* Click on Save.



* Click on Build Now which is in the left navigation menu.
* Building of a job takes some time. After building, if it is green then it is success.
* Otherwise if it red then it is failed.
* When your job is built successfully, then a HTML Report is generated.



* Click on HTML Report to display your Web page as output. Make sure all the required chrome extensions have access to your output page. If access is on requesting then select them to give access.
* Your output screen will look like this.



**Fig 2.2.5. Implementing Web Page through Jenkins Job**

## Implementation of Jenkins Pipelines:

### What is a pipeline in Jenkins ?

Jenkins is a popular open-source automation server that allows you to automate various aspects of the software development and deployment process. Jenkins pipelines are a way to define and automate your continuous integration and continuous delivery (CI/CD) workflows within Jenkins. Jenkins pipelines provide a way to represent your build, test, and deployment processes as code, which makes it easier to version, manage, and share your automation workflows.

There are two main types of pipelines in Jenkins:

### Scripted Pipeline:

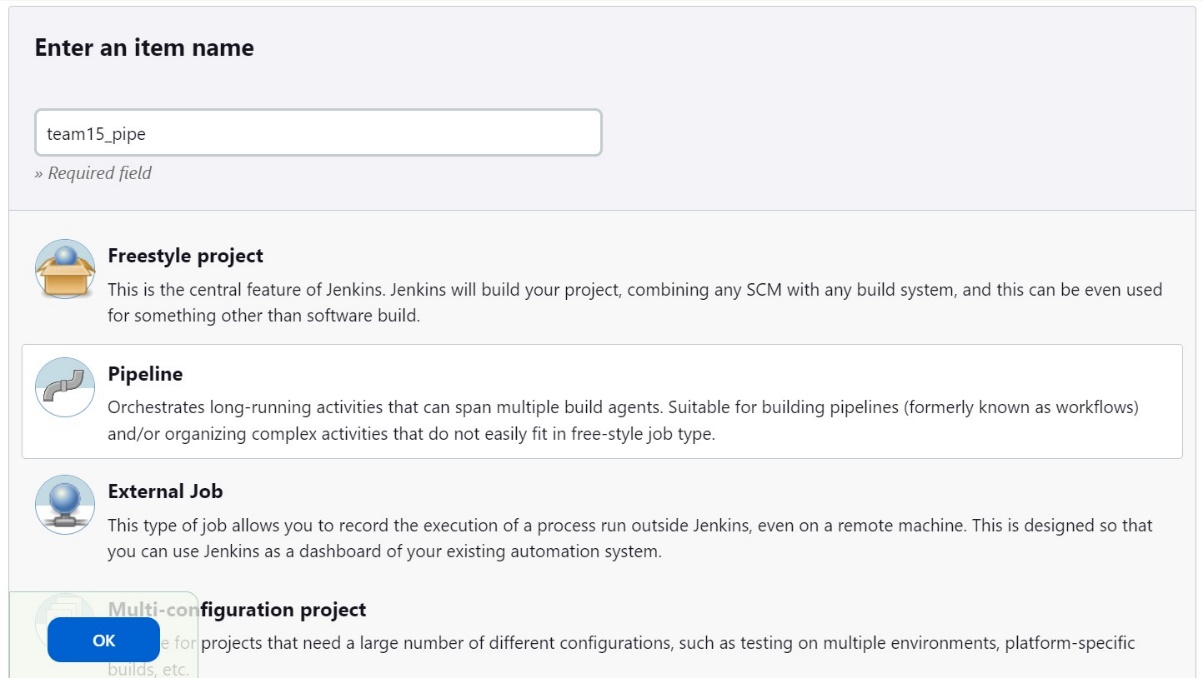
This is the traditional way of defining Jenkins pipelines using Groovy-based DSL (Domain Specific Language). Scripted pipelines are written in a Groovy script and provide great flexibility, but they can be complex to manage and maintain.

### Declarative Pipeline:

Declarative pipelines are a more structured and human-readable way to define your CI/CD workflows. They are defined using a simpler syntax and are recommended for those who want to get started quickly and maintain pipelines more easily.

### Deploymemt of a job using Declarative Pipeline:

* Open Jenkins Dashboard. Click on New Item.
* Enter a name for Pipeline. Select Pipeline and Click OK.



* Scroll down until you see Pipeline, type the following in the Script Box.

**pipeline {**

**agent any**

**stages {**

**}**

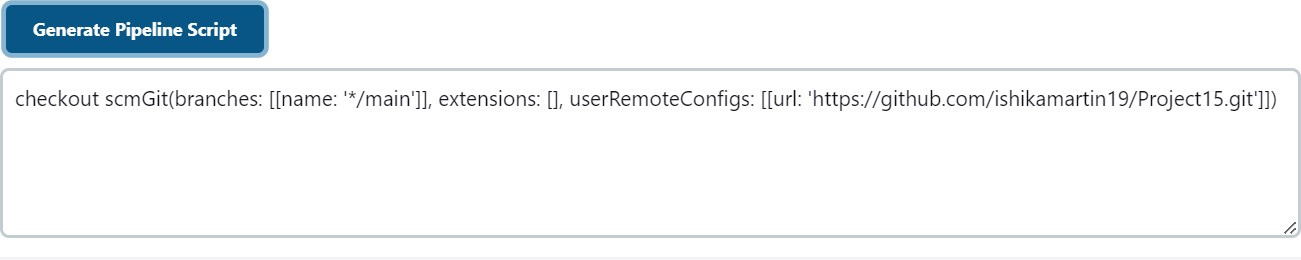
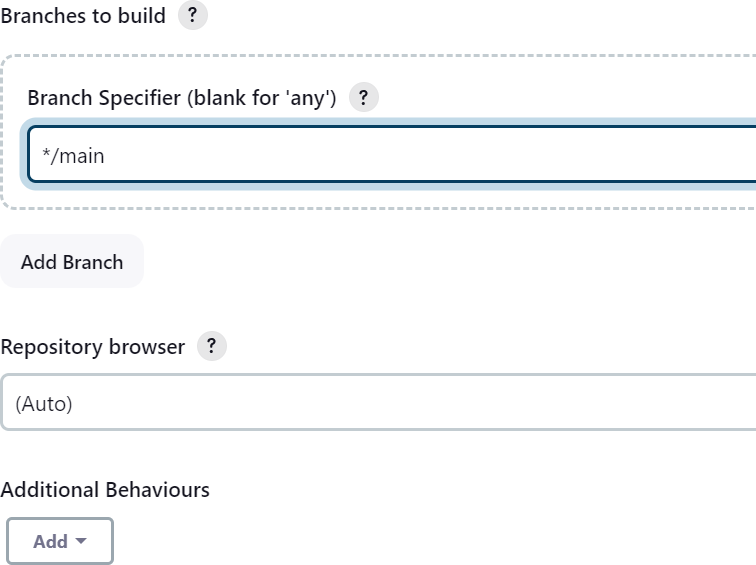
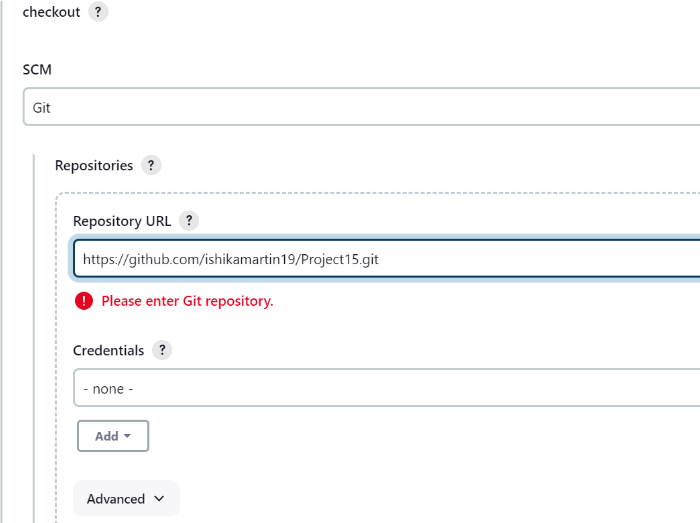
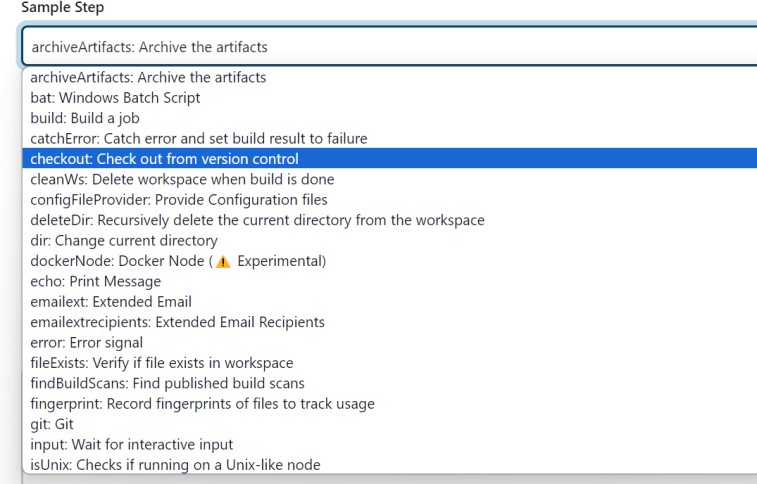
**}**



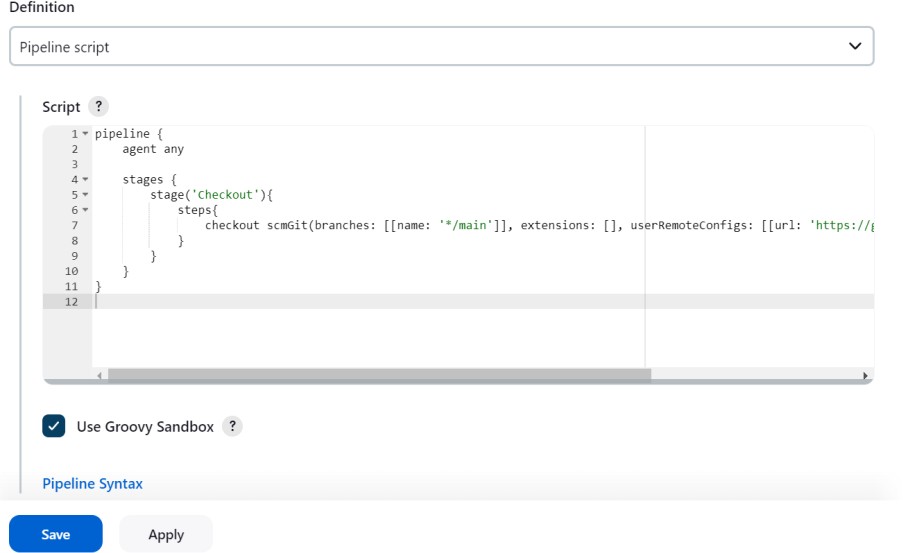
* Now inside stages, we create different stages which are executed for their respective purposes. We must make sure that each stage name should be unique to avoid failed build.

### Stage-1: checkout

Click on Pipeline Syntax which is just below the Script Box. In Sample Step dropdown, Select checkout, paste the Repository URL and branch name, Click on Generate Pipeline Script which is below the Sample Step.



* After clicking on Generate Pipeline Script, you will get a statement. Copy paste the statement inside Checkout stage in the Script.

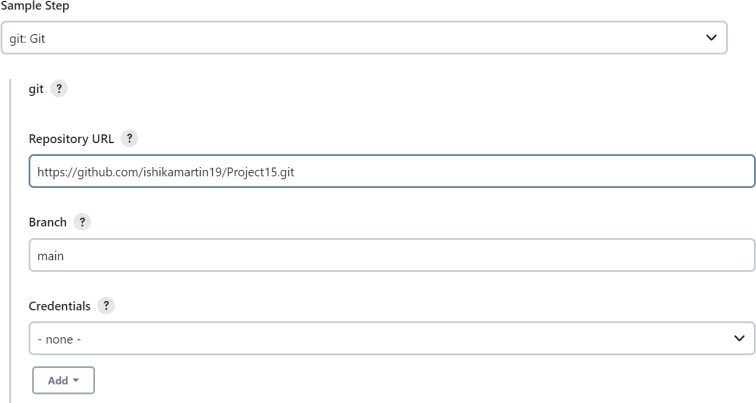


### Stage-2: git

As we are building a project which is pushed into GitHub, we need git pipeline to connect our project from Git to Jenkins.

Click on Pipeline Syntax which is just below the Script Box. In Sample Step dropdown, Select git, paste the Repository URL and branch name, Click on Generate Pipeline Script which is below the Sample Step.

* After clicking on Generate Pipeline Script, you will get a statement. Copy paste the statement inside Git stage in the Script.



### Stage-3: publishHTML

As we are building a HTML web page in Jenkins, publishHTML pipeline is important to get the output of our project.

Click on Pipeline Syntax which is just below the Script Box. In Sample Step dropdown, Select publishHTML, paste the project folder name as HTML directory and first introduction page name as Index page, Click on Generate Pipeline Script which is below the Sample Step.

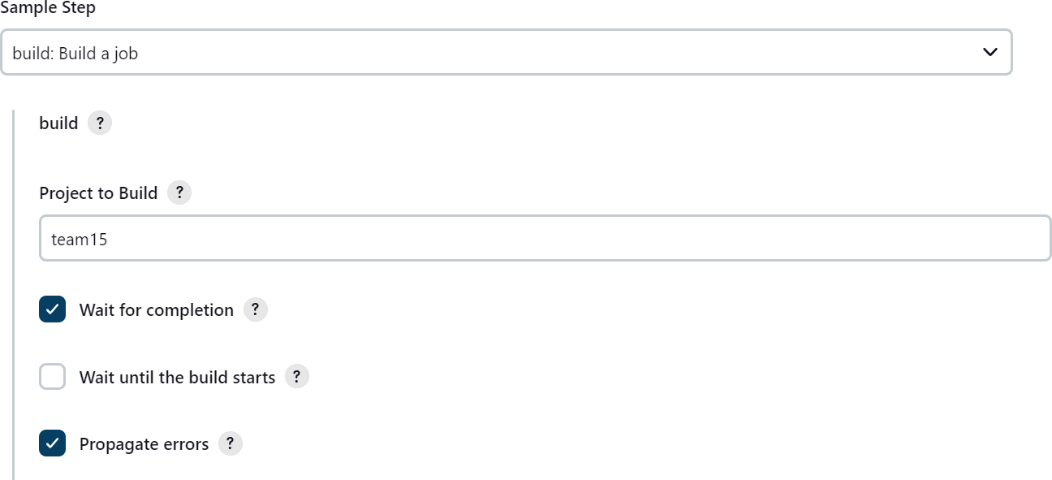
* After clicking on Generate Pipeline Script, you will get a statement. Copy paste the statement inside HTML stage in the Script.



### Stage-4: build

Click on Pipeline Syntax which is just below the Script Box. In Sample Step dropdown, Select build, Give the name of the job which is already built to deploy our webpage **team15**, Click on Generate Pipeline Script which is below the Sample Step.

* After clicking on Generate Pipeline Script, you will get a statement. Copy paste the statement inside Build stage in the Script.



### post section in declarative pipeline:

The **post** section is used to define actions that should be taken after the pipeline has executed. In this example, there are three post-condition blocks: success, failure and always. You can include different actions or notifications in each block based on the outcome of the pipeline. **Code for declaring post conditions:**

post{

success{

echo('This will be printed if the build is success')

}

failure{

echo('This will be printed if the build is failure')

}

always{

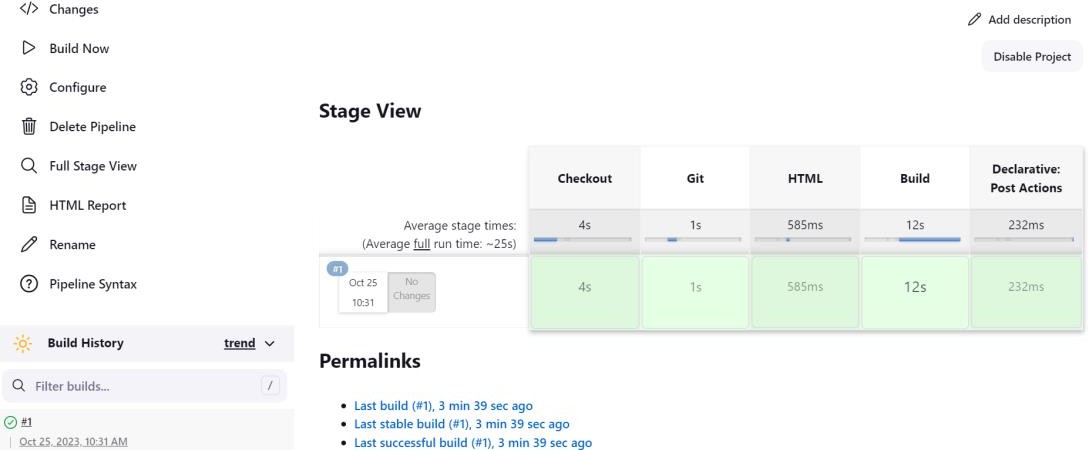
echo('This will be printed even if the build is failure')

}

}

The above code is written after closing stages.

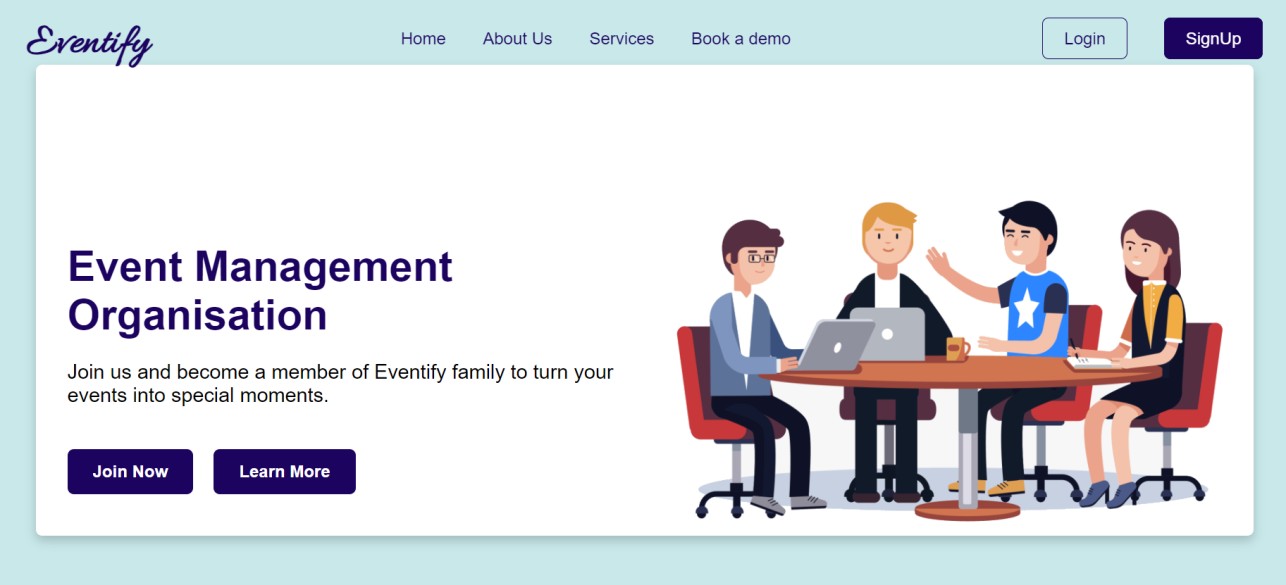
* After writing the entire code, save and click on Build Now.
* It takes time to build as it consists of various stages and it executes each stage seperately including the post section.
* After build is completed, if all stages are green it means that the entire building of pipeline is success.
* Otherwise, you must go to configure again, cross check the errors and build again until your pipeline is successfully build.



* Click on HTML Report to view the output of your webpage. Make sure that all the extensions and plugins have access inorder to avoid content blocking.

## Results:

### Home page:

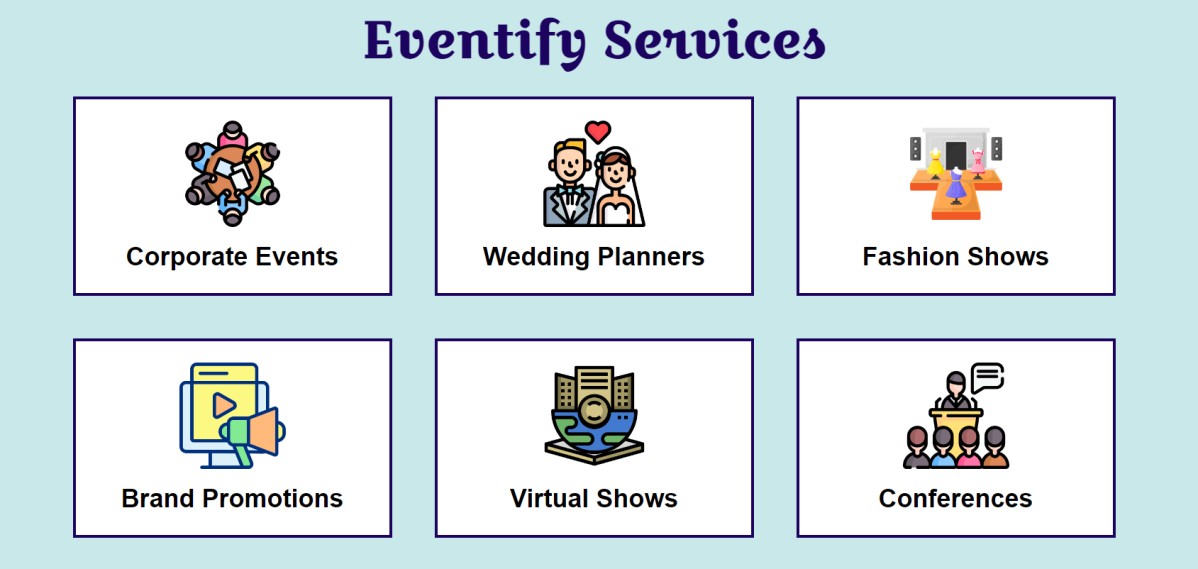


**Fig 2.2.7.1 Implementing Web Page through Jenkins Pipeline**

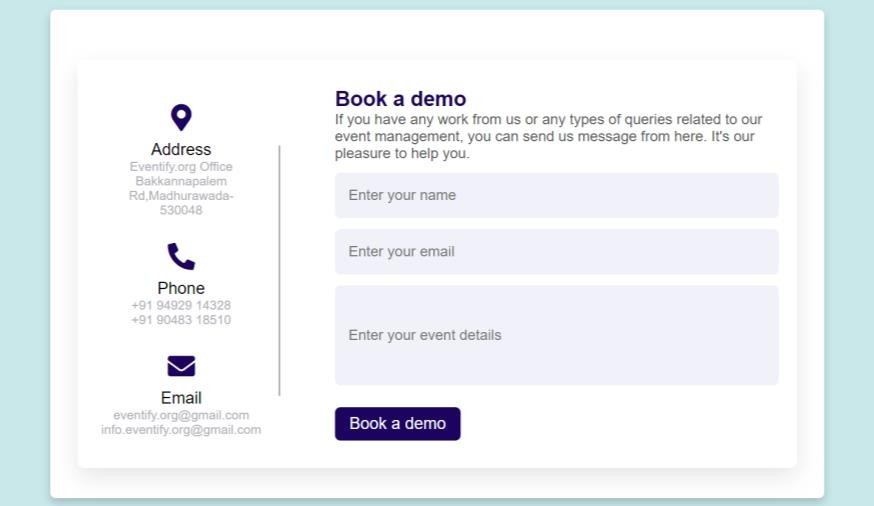
### About Us page:



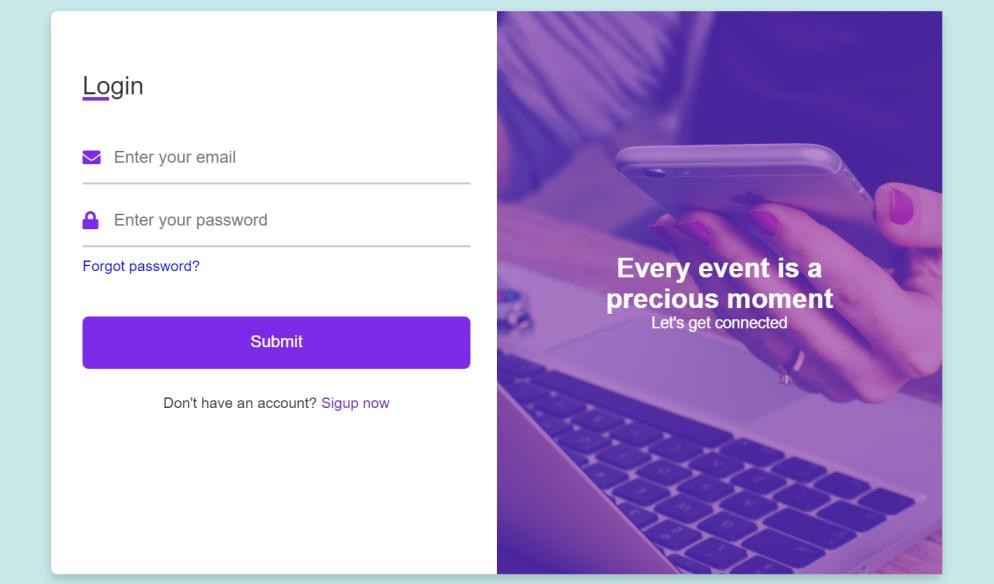
* + - 1. **Services page:**



### Book a demo page:



* + - 1. **Login and Sign Up page:**



# CONCLUSION

In conclusion, implementing a Continuous Integration and Continuous Deployment (CI/CD) system using Jenkins, coupled with an event management system in HTML, can be a highly effective approach to streamline software development and delivery processes. This combination offers several benefits:

1. **Automation:** Jenkins provides powerful automation capabilities for building, testing, and deploying software, while HTML-based event management systems allow for easy customization and automation of event-triggered actions. This reduces manual intervention and minimizes human errors.
2. **Collaboration:** CI/CD using Jenkins fosters collaboration among development, testing, and operations teams by providing a centralized platform for managing code, sharing artifacts, and coordinating deployments. An HTML event management system can enhance this collaboration by offering real-time event tracking and communication.
3. **Speed and Efficiency:** With CI/CD, software can be built, tested, and deployed more rapidly, leading to faster releases and quicker response to customer feedback or market demands. Event management systems help in monitoring the progress of these processes and responding to issues promptly.
4. **Quality Assurance:** Continuous testing and automated quality checks in Jenkins help maintain the overall quality of the software. Event-driven HTML systems can capture and track issues, allowing for prompt resolution and quality improvement.
5. **Scalability:** Jenkins and HTML-based event management systems are highly scalable, accommodating the needs of both small and large development teams and projects. As the software development environment grows, this combination can adapt accordingly.
6. **Transparency:** By integrating an event management system into the CI/CD pipeline, teams gain visibility into the entire software development and delivery process. Stakeholders can monitor progress and track key events in real time, ensuring better transparency and accountability.
7. **Flexibility:** HTML-based event management systems can be customized to suit the specific needs and preferences of your organization, making it easier to adapt the system to unique workflows and event handling requirements.

Combining Jenkins for CI/CD with an HTML-based event management system is a powerful strategy for enhancing software development and deployment processes. This approach promotes automation, collaboration, speed, quality assurance, scalability, transparency, and flexibility. It empowers organizations to keep up with the rapid pace of software development and maintain a competitive edge in the ever-evolving tech landscape.