UNC CHARLOTTE
College of Computing and Informatics

DEPARTMENT OF COMPUTER SCIENCE
ITCS 5102 SURVEY OF PROGRAMMING LANGUAGES

Group Members

Rachana Gullipalli	- 801311637
Avinash Koya	- 801312690
Sahithi Paladugu	- 801327324
Manasa Avula	- 801307493
Ramu Sirusanagandla	- 801329345
Mahija Shivani Dadisetty	- 801308384

Table of Contents:

Introduction to Go Programming Language:

A well-liked open-source programming language created by Google in 2009 is called GoLang, also known as Golang. It rapidly became a popular among developers since it was created to be straightforward, effective, and simple to use. Due to the fact that GoLang is compiled, it is quickly and effectively transformed into machine code before use. Concurrency support is one of the primary characteristics of GoLang. GoLang provides built-in concurrent programming capabilities. Concurrency is the ability of a program to carry out many activities at once.

It is an excellent choice for implementing huge distributed systems which can handle concurrent applications as a result. The simplicity and ease of implementation makes GoLang a very popular language.The robust standard library with many practical tools, including support for network programming, cryptography, and web development makes it an ideal language to use.

Paradigm of the Language:

The core programming paradigm of a certain programming language describes the types of programming that it enables. Object-oriented programming (OOP) ideas like encapsulation, inheritance, and polymorphism are supported in GoLang, a procedural programming language that is primarily used.

GoLang supports feature like closures, higher-order functions, and first-class functions. This indicates that the language’s functions can be used as variables orelse can be modified like any other data type. The concurrency support that is provided by this language is the best.

The concurrency support that is provided by this language has made it simple for providing high-concurrency application development in GoLang, such as channels and goroutines. Goroutines are lightweight threads that enable the execution of numerous tasks concurrently, whereas channels are used to communicate across concurrent processes.

Historical Evolution:

- Robert Griesemer, Rob Pike, and Ken Thompson started working on the GoLang project in 2007 at Google.
- GoLang made its first stable release in March 2012 after being made publically known in November 2009.
- GoLang was inspired by a number of different programming languages, including C, Pascal, and Oberon.
- Its original purpose was to produce a language that was simpler to use for concurrent programming than already-existing languages like C++ and Java.
- GoLang contains a garbage collector that automatically handles memory allocation and deallocation, making it simpler to write dependable code.
- GoLang was created to be a compiled language with quick compilation times, making it appropriate for large-scale development projects.
- GoLang has built-in concurrency support, including goroutines and channels, which makes it simpler to write concurrent code that is free of race conditions and deadlocks.
- GoLang has a strong focus on simplicity, readability, and maintainability, making it a popular choice for both open source and commercial software development.
- As of 2021, version 1.17 of GoLang is the most up-to-date stable release, and it comes with a number of changes and new features.

Why use Go?

- Fast and Efficient: GoLang is a compiled language that generates code that is both quick and effective.
- Concurrency: GoLang has built-in concurrency support, making it simple to develop highly parallel code without the risk of race situations or deadlocks. GoLang is a great choice for creating distributed systems and other highly concurrent applications because of this characteristic.
- Simplicity: GoLang is simple to learn and use, with a basic syntax. For novices who wish to learn programming without the complexities of other languages, its minimalist approach to programming makes it a fantastic alternative.
- Scalability: Because GoLang is built to fulfill the scalability requirements of contemporary software systems, it is a great option for creating complex applications.
- Standard Library: GoLang has a strong standard library that provides a lot of helpful features and functions for networking, web development, and other things.
- Cross-Platform Support: GoLang works with a variety of operating systems, including Windows, macOS, Linux, and FreeBSD.
- Open-Source Community: There is a burgeoning developer and contributor community for GoLang, and there are lots of third-party libraries and tools

available for use. As a result, getting support and assistance when creating GoLang applications is simple.

- Security: GoLang comes with a number of features that enhance application security, such as support for secure communication protocols and strong typing.

Elements Of the Language:

GoLang has several key elements that make it a powerful and efficient programming language.

Data types in Go:

- **Integer Data types:**

Data type	**Size (bits)**	**Range**
int	platform	2^31 to 2^31-1 (32-bit) or -2^63 to 2^63-1 (64-bit)
int8	8	-128 to 127
int16	16	-32768 to 32767
int32	32	-2147483648 to 2147483647
int64	64	-9223372036854775808 to 9223372036854775807
uint	platform	0 to 2^32-1 (32-bit) or 0 to 2^64-1 (64-bit)
uint8	8	0 to 255
uint16	16	0 to 65535
uint32	32	0 to 4294967295
uint64	64	0 to 18446744073709551615

Whole numbers are represented using Integers. GoLang provides a wide range of integer datatypes with differing the sizes and ranges to suit multiple use cases.

Example:

```
package main
import "fmt"
func main() {
   var x int = 42
   fmt.Println(x)
}
```

In this example, we declare and initialize an integer variable x with a value of 42.

- **Float Data Types:**

Data Type	**Size (bits)**	**Range**
float32	32	-3.4E38 to 3.4E38 (about 7 decimal digits of precision)
float32	64	-1.7E308 to 1.7E308 (about 15 decimal digits of precision)

float data types included are:
float32 and float64.
float32: recession upto 7 digits
float64: the precision is 15 decimal digits.

Example :

```
package main

import "fmt"

func main() {
   var x float64 = 3.14159
   fmt.Println(x)
}
```

In this example, we declare and initialize a float variable x with a value of 3.14159.

- **Complex Data Types:**

Data Type	**Size (bits)**	**Range**
complex64	**64**	**Real and imaginary parts are float32**
complex128	**128**	**Real and imaginary parts are float64**

Complex numbers are used to represent numbers with both real and imaginary parts. GoLang provides two complex data types: complex64 and complex128. In the complex64 data type the complex number with float32 real and imaginary parts, and the complex128 data type is a complex number with float64 real and imaginary parts.

Example :

```
package main
import "fmt"
func main() {
   var x complex128 = 1 + 2i
   fmt.Println(x)
}
```

In this example, we declare and initialize a complex variable x with a value of 1 + 2i.

- **Boolean Data Type:**

Data Type	**Size (bits)**	**Range**
bool	1	true or false

Boolean values: these are used to represent the logical values, such as true or false. GoLang provides a single Boolean data type: bool.

Example:

```
package main
import "fmt"
func main() {
   var x bool = true
   fmt.Println(x)
}
```

In the above example, the Boolean value ‘true’ we are initializing to the variable ‘x’

- **String Data Type:**

Data Type	**Size (bits)**	**Range**
string	platform	A sequence of bytes representing a text string (UTF-8)

Strings are used to represent text data. GoLang provides a single string data type: string. Strings are represented as a sequence of bytes representing a text string encoded in UTF-8 format.

Example:

```
package main
import "fmt"
func main() {
    var x string = "Welcome to GoLang!"
    fmt.Println(x)}
```

Here, we are declaring and initializing the variable ‘x’ to the string "Welcome to GoLang!"

Syntax of the Go Language:

Go Variables:

Naming convention in Golang:

A variable name must begin with a letter or an underscore (_).
Letters (A-Z, a-z), numbers (0–9), and underscores (_) are all acceptable characters for variable names.
Variable names are case sensitive.

- Keywords cannot be used as variable names.
- Variable names should be kept short and meaningful.

There are two ways to declare variables in Go:

Using var keyword:
Variables can be created in Go using the var keyword. Here's the syntax:
var variable_name type = expression

Short Variable Declarations:
Short variable declarations are used to declare and initialize local variables in Go functions. Here's the syntax:
variable_name := expression

Go Operators:
For performing arithmetic, relational, logical, and bitwise operations, Go offers a number of operators. The most popular Go operators are listed below:

Arithmetic Operators:
Arithmetic operators are used to perform mathematical operations on operands. Here are the arithmetic operators in Go:

- Addition (+): Adds two operands.
- Subtraction (-): Subtracts two operands.
- Multiplication (*): Multiplies two operands.
- Division (/): Divides the first operand by the second operand.
- Modulus (%): Returns the remainder of dividing the first operand by the second operand.

Example:
a = 10
b = 5
c = a + b => c = 15
d = a - b => d = 5
e = a * b => e = 50
f = a / b => f = 2
g = a % b => g = 0

Relational Operators:
Relational operators are used to compare two operands. Here are the relational operators in Go:

- Equal to (==): If the two operands are equal it returns as true.
- Not equal to (!=): If the two operands are not equal it returns as true.
- Greater than (>): If the first operand is greater than the second operand it returns as true.
- Less than (<): If the first operand is less than the second operand it returns as true.
- Greater than or equal to (>=): If the first operand is greater than or equal to the second operand it returns as true.
- Less than or equal to (<=): If the first operand is less than or equal to the second operand it returns as true.

Example:
a = 10
b = 5
c = 10
d = a == b => d = false
e = a != b => e = true
f = a > b => f = true
g = a < b => g = false
h = a >= c => h = true
i = b <= c => i = true

Logical Operators:
Logical operators are used to perform logical operations on boolean values. Here are the logical operators in Go:

- Logical AND (&&): If both expressions are true it returns true.
- Logical OR (||): If at least one expression is true it returns true.
- Logical NOT (!): Returns the opposite of the expression's value.

Example:
a = true
b = false
c = true
d = a && b => d = false

e = a || b => e = true
f = !c => f = false

Bitwise Operators:
Bitwise operators are used to perform bitwise operations on two operands. Here are the bitwise operators in Go:

- & (bitwise AND): Performs an AND operation on each bit of the operands. 1 is the outcome if both bits are 1. If not, the outcome is 0.
- | (bitwise OR): Performs an OR operation on each bit of the operands The outcome is 1 if either bit is 1. If not, the outcome is 0.
- ^ (bitwise XOR): Performs an exclusive OR (XOR) operation on each bit of the operands. When the bits vary, the outcome is 1. If not, the outcome is 0.
- << (left shift): Shifts the bits of the first operand to the left by the number of places specified by the second operand. Zeros are shifted in from the right.
- >>(right shift): Shifts the bits of the first operand to the right by the number of places specified by the second operand. Zeros are shifted in from the left (if the number is unsigned) or the sign bit (if the number is signed).
- &^ (AND NOT): Performs a bitwise AND of the first operand and the complement of the second operand. This is sometimes called a "bit clear" operation, since any bits that are 1 in the second operand are cleared (set to 0) in the result.

Assignment Operators:

Assignment operators in GoLang are:

- *= : This operator multiplies the right-hand side value with the left-hand side value and then assigns the result to the left-hand side variable.
- /= : This operator divides the left-hand side value by the right-hand side value and then assigns the result to the left-hand side variable.
- %= : This operator calculates the remainder when the left-hand side value is divided by the right-hand side value, and then assigns the result to the left-hand side variable.
- Similarly we have it for the addition and subtraction as : += and -= respectively.

Misc Operators:

- &: The (&) operator returns the memory address of a variable.
- *: The (*) operator is used to declare a pointer variable or to dereference a pointer variable.
- <-: The arrow (<-) operator is used in Go to send and receive values from channels. It is called as "receive operator" and is used to retrieve a value from a channel.

Control Statements:
Control statements are an essential part of programming languages that help to execute specific blocks of code based on specific conditions.
There are different types of if-else expressions in Go:

- if expression
- if-else expression
- if-else-if ladder expression
- nested if expression

if expression: When a block code meets the requirement only then will the control be shifted to work on another piece of code. This is how the 'if' control statement works.

Syntax:
```
if condition {
  // Statements to execute if condition is true
}
```
if-else expression: This case is when if a particular condition is met then control goes to one piece of code else it shifts and executes another block of code.

Syntax:
```
if condition {
  // code
}
else {
  // code
}
```
if-else-if ladder expression: When several conditions need to be tested, this expression is used. The program advances to the next condition if the first one is false, and so on. If none of the requirements are true, the last else block will be carried out.
Syntax:
```
if(condition1) {
  // code
}
else if(condition2) {
  // code
}
else{
  // code
}
```

nested if expression: When one if statement is nested inside another if statement, this expression is used. The nested if block's code will run if the first condition is satisfied. The code inside the second condition (inside the nested if block) will likewise run if the second condition is true, and so on.

Syntax:

```
if(condition1) {
  // code 1
  if(condition2) {
    // code 2
  }
}
```

Loops:

Simple for loop:

These are the basic for loop structure that can be used for looping.

Syntax:

```
for initialization; condition; post {
// statements to execute repeatedly
}
```

Infinite for loop:

The for loop can be made limitless by eliminating all three expressions. The loop will keep running until the program is stopped.

Syntax:

```
for {
// statements to execute repeatedly
}
```

While loop using for:

A while loop can be implemented using a for loop. The loop will execute repeatedly until a specified condition is no longer true.

Syntax:

```
for condition {
// statements to execute repeatedly
}
```

Simple range in for loop:

To run in loops , be it objects in an array, slice, or map, we use a range keyword in a for loop. Each item's value and index are both returned.

Syntax:

```
for i, j := range iterable {
// statements to execute for each item
}
```

For loop for strings:

The Unicode code points of a text can be iterated over using a for loop. This gives you the ability to change specific characters within a string.

Syntax:

```
for index, char := range str {
// statements to execute for each character
}
```

For loop for maps:

A map's key-value pairs can be iterated through using a for loop.

Syntax:

```
for key, value := range myMap {
// statements to execute for each key-value pair
}
```

For loop for channels:

Up until the channel is closed, values sent on it can be iterated over in a for loop.

Syntax:

```
for item := range myChannel {
// statements to execute for each item
}
```

Switch Statement:
Go language supports two types of switch statements: expression switch and type switch.

Expression Switch: It allows allocating execution to various code segments based on the expression's value.
The syntax of an expression switch is as follows:

```
switch optstatement; optexpression {
case expression1: Statement
case expression2: Statement
default: Statement
}
```

optstatement: It is an optional statement that is executed before the switch expression is evaluated.
optexpression: It is an optional expression that is evaluated once.
case expression: It is the value that will be compared with the switch expression. If the case expression matches the switch expression, the corresponding statements will be executed.
default: It is executed when none of the case expressions matches the switch expression.

Type Switch: It is used when comparing types. The type that will be compared with the type in the switch expression is contained in the case.
The syntax of a type switch is as follows:

```
switch optstatement; typeswitchexpression {
case typelist 1: Statement
case typelist 2: Statement
default: Statement
}
```

optstatement: It is an optional statement that is executed before the switch expression is evaluated.
typeswitchexpression: It is an expression that is evaluated once, and its result must be of interface type.
typelist: A list of kinds that will be compared to the type in the switch expression is contained there. The appropriate statements will be performed if the type in the

switch expression matches any of the types in the typelist.
default: It is executed when none of the types in the typelist matches the type in the switch expression.

Functions:
Go's fundamental concept of functions enables programmers to write code segments or statements that can be reused throughout a program.

Syntax :

```
func function_name(parameters)(return_type){
 // function body
}
```

function_name: This is the name of the function.
parameters: Comma-separated, it is a list of optional input options.
return_type: The optional value's type, which the function returned, is what it is.
body component: The code or instructions that will be executed are contained in the function when it is called.

Main() Function:

This is a special kind of function that doesn't take any input and outputs nothing. In Go, the main() method does not need to be explicitly called; instead, each executable program needs to have a single main package and main() function. The main() method is called by Go automatically.

Syntax:

```
package main
import "fmt"
func main() {
    // code to be executed
}
```

Init() Function:

The init() function, here we declare the global variables. The many functions declared are executed in the order they are declared.

Syntax:

```
func init() {
   // code to initialize global variables
}
```

Pros and Cons of Golang Programming Language:

Pros of the Language:

- Efficiency: GoLang is fast and efficient, making it an excellent choice for large-scale, distributed systems and highly concurrent applications.
- Concurrency: GoLang has built-in support for concurrency, which makes it easy to write highly concurrent code without the risk of race conditions or deadlocks.
- Simplicity: GoLang has a simple syntax and is easy to learn and use, which makes it an excellent choice for beginners.
- Standard Library: GoLang has a robust standard library that includes many useful functions and features for networking, web development, and more.

Cons of GoLang include:

- Limited OOP Support: GoLang has support for object-oriented programming concepts, but it is not a pure OOP language.
- Lack of Third-Party Libraries: Compared to other languages such as Python and Ruby, GoLang has a smaller community and fewer third-party libraries and tools.
- Lack of Generics: GoLang does not have support for generics, which can make certain programming tasks more challenging.

Why is it better than other languages?

GoLang is a powerful and efficient programming language.

- Efficiency: GoLang is fast and efficient, making it an excellent choice for large-scale, distributed systems and highly concurrent applications.
- Concurrency: GoLang has built-in support for concurrency, which makes it easy to write highly concurrent code without the risk of race conditions or deadlocks.
- Simplicity: GoLang has a simple syntax and is easy to learn and use, which makes it an excellent choice for beginners.

- Standard Library: GoLang has a robust standard library that includes many useful functions and features for networking, web development, and more.
- Scalability: GoLang is designed to handle the scalability requirements of modern software systems, making it an excellent choice for building large-scale applications.

How the language made the programs easy/hard to implement?

The language we have selected could influence how simple or difficult it is to implement programs. For example, several features are present in Go Lang which makes implementation simple. One of these features is simplicity in the programming language which makes debugging and code review simple. Go Lang has a slim library design which makes debugging easy and it makes deployments easy. Another aspect that makes it easy to implement in GO Lang is static typing. Despite being static, it doesn't stop the development of the code. Because of the large number of software development kits (SDKs), and open-source libraries, cloud service makes it simple for us to develop complex features. The testing capabilities of Go are commendable so it makes team development easier.

Challenges Faced while Programming:

We have faced several challenges during our development process using the go language. On such challenge is various ways a task is to be performed. Go lang does provide multiple ways to complete tasks which are various declaration syntaxes and return variables which leads to issues while performing development. Other challenges we faced are Go Lang’s lack of documentation and libraries.

Go Lang has fewer libraries available than other well-known programming languages like Perl and Python, which are more widely used. Go can become more efficient and competitive by using "import." instead of identifying each reference to an imported function by the package it is imported from. Additionally using pointers in Go Lang can be challenging for our team because it is a new language. Incorrect usage of pointers leads to runtime errors and bugs. So, it is necessary for us to allocate time to learn about the usage of pointers.

Problem statement

It is true that universities do not offer a specific portal where graduates can apply for jobs straight after graduation. Career fairs are frequently held on campuses so that students can speak with possible companies and discover work prospects. A student may lose a great opportunity if he misses the career fair due to some other important reasons.

Furthermore, even if a student shows up at the career fair, there is no guarantee that the recruiter will get their resume or that they will be given consideration for the job. This can be frustrating for both the recruiter and the student because time and money are wasted during the recruitment process.

It would be more simple and simplified for both students and recruiters if there were a specific career portal for students. Without needing to go to a particular event, students could search and apply for jobs whenever and from wherever they were. The pool of prospects that recruiters could possibly reach would grow as a result of the increased efficiency with which they could analyze resumes and applications.

Overall, launching a special job platform for students would be a great idea because it would benefit both sides and make the hiring process much easier.

Proposed Method

The Proposed student job portal will have two portals one for the student and another for the recruiter. There will be a common login and signup page for both students and recruiters. Based on the credentials the person is navigated to respective portals. If student is logged in, he can see a page where he can view jobs that are available, the jobs he has applied already and also a search bar is present where he can search for a specific company or role he is looking for. When it comes to recruiter portal the recruiter can add new job roles in his company and view student's details who have applied for a position in his company.

Experiments

The experiments have been done on our job portal to be certain that it helps to reduce the gap between the job seeker that is the student and the recruiter. Any student or any recruiter can sign up in the portal and start using the portal for either applying or recruiting for job positions. As separate portal is designed for both recruiters and students which will be directed based on the login credentials.

A recruiter can add a job role so the students can apply to it. The students can login into their portal and see the available jobs. Once the student sees the position and thinks it is ideal for them, they can start uploading their resumes. All the applied job positions by the student are visible in the applied jobs page. The students can also search for jobs by using the search bar at the top if they are looking for a specific company or role to apply.

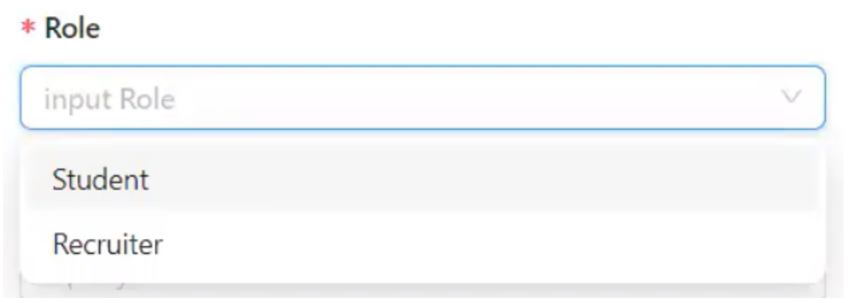
The recruiter can see who applied to the positions in his portal. He can review the resumes and directly contact the student.

Features Highlighted

Our team has developed a Job recruiting portal due to the lack of a dedicated portal for students to apply for jobs directly after graduation and also due to the challenges of attending career fairs. Here are the highlighted features we have developed:

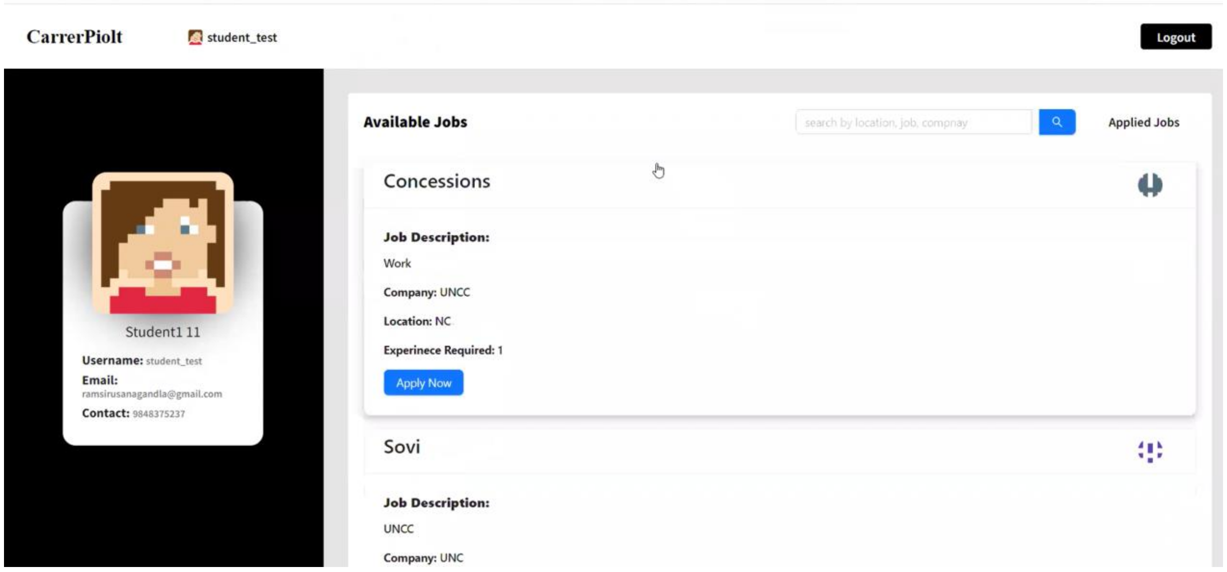
Login system:

This website has two different logins one is a student login another one is a recruiter login this helps to ensure the right person has access to appropriate features. On our website we have created a field (Role) where we can classify you are a recruiter or a student to classify the access.



Student functionality:

The website allows students to look for job vacancies, submit applications for companies they are interested in, and upload their resumes. Students now have direct access to a wide range of job opportunities while also streamlining the job application process.



Student1 11

Username: student_test

Email: ramsirusanagandla@gmail.com

Contact: 9848375237

Concessions

Job Description:

Work

Company: UNCC

Location: NC

Experinece Required: 1

Sovi

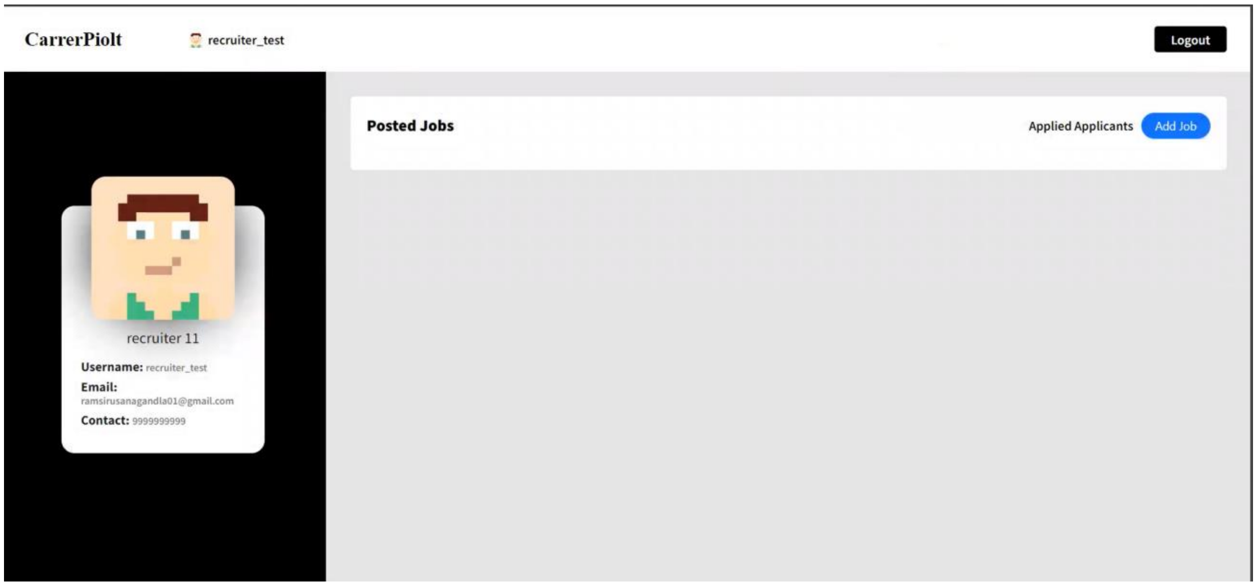
Job Description:

UNCC

Company: UNC

Recruiter functionality:

Recruiters can login and view the resumes of candidates who have submitted applications for open positions. Employers may easily identify potential employees thanks to this, which enables targeted recruiting.



recruiter 11

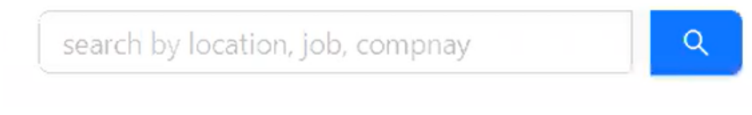
Username: recruiter_test

Email: ramsirusanagandla01@gmail.com

Contact: 9999999999

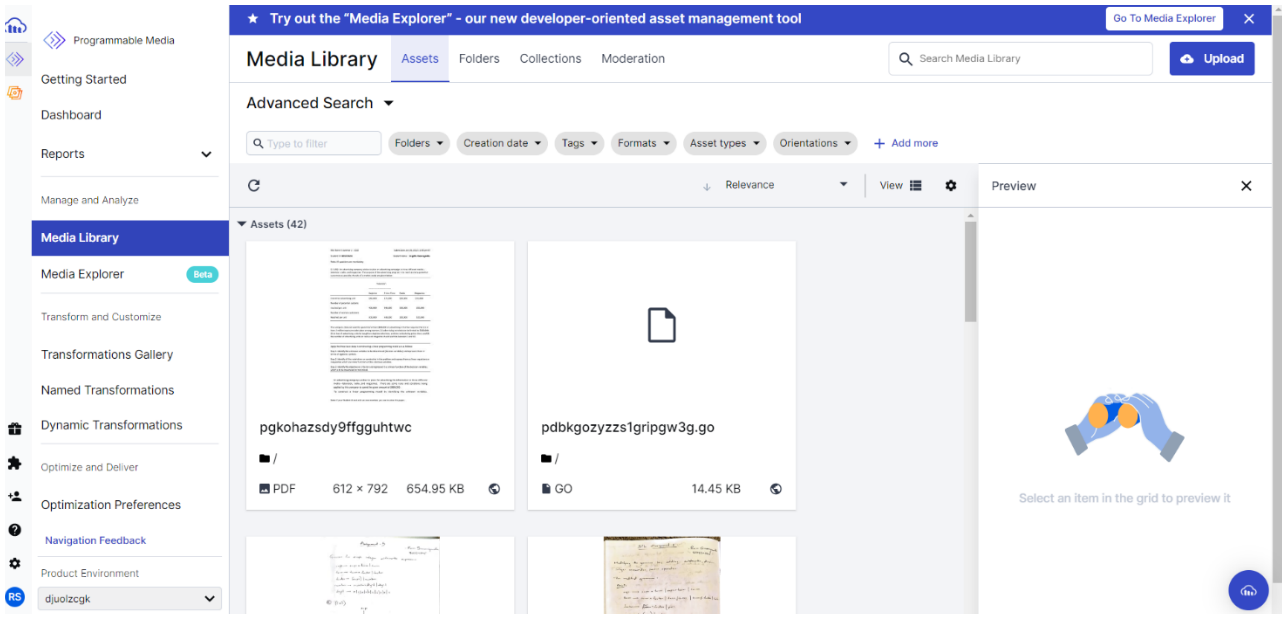
Search functionality:

The search box makes it simple to navigate and filter job openings based on location, job, and company. As a result, Users may find it quicker and simpler to identify appropriate job openings as a result.



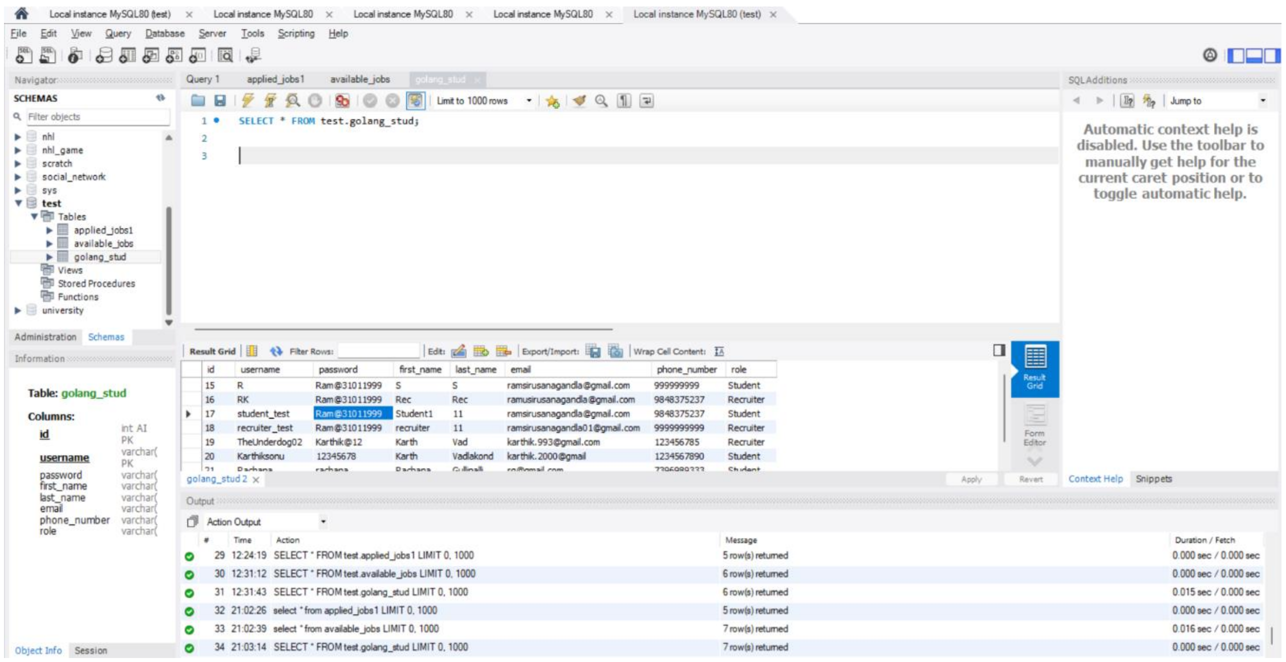
Cloudinary:

Cloudinary is a free platform and been used to store the resumes of the students and this stored resume link will be given to recruiter. Resumes stored in cloudinary will only be visible to the owner of that account but not by recruiters or students.



Database: MySQL Workbench 8.0:

For the database we used MySQL Workbench 8.0 for storing the recruiter and the student details. This comprises of three tables namely: goland_stud, this stores the login details that were captured during signup. We also have the applied_jobs1 table which stores the application details of the candidates who applied for jobs. The available_jobs table has the details of the data that the recruiter posts regarding a particular job.



```
SELECT * FROM test.golang_stud;
```

id	username	password	first_name	last_name	email	phone_number	role
15	R	Ram@31011999	S	S	ramsirusanagandla@gmail.com	999999999	Student
16	RK	Ram@31011999	Rec	Rec	ramusirusanagandla@gmail.com	9848375237	Recruiter
17	student_test	Ram@31011999	Student1	11	ramsirusanagandla@gmail.com	9848375237	Student
18	recruiter_test	Ram@31011999	recruiter	11	ramsirusanagandla01@gmail.com	9999999999	Recruiter
19	TheUnderdog02	Karthik@12	Karth	Vad	karthik.993@gmail.com	123456785	Recruiter
20	Karthiksonu	12345678	Karth	Vadlakond	karthik.2000@gmail	1234567890	Student
21	Rachana	rachana	Rachana	Gullipalli	rg@gmail.com	7396989333	Student

```
#   Time      Action                                          Message           Duration / Fetch
29  12:24:19  SELECT * FROM test.applied_jobs1 LIMIT 0, 1000   5 row(s) returned  0.000 sec / 0.000 sec
30  12:31:12  SELECT * FROM test.available_jobs LIMIT 0, 1000  6 row(s) returned  0.000 sec / 0.000 sec
31  12:31:43  SELECT * FROM test.golang_stud LIMIT 0, 1000     6 row(s) returned  0.015 sec / 0.000 sec
32  21:02:26  select * from applied_jobs1 LIMIT 0, 1000        5 row(s) returned  0.000 sec / 0.000 sec
33  21:02:39  select * from available_jobs LIMIT 0, 1000       7 row(s) returned  0.016 sec / 0.000 sec
34  21:03:14  SELECT * FROM test.golang_stud LIMIT 0, 1000     7 row(s) returned  0.000 sec / 0.000 sec
```

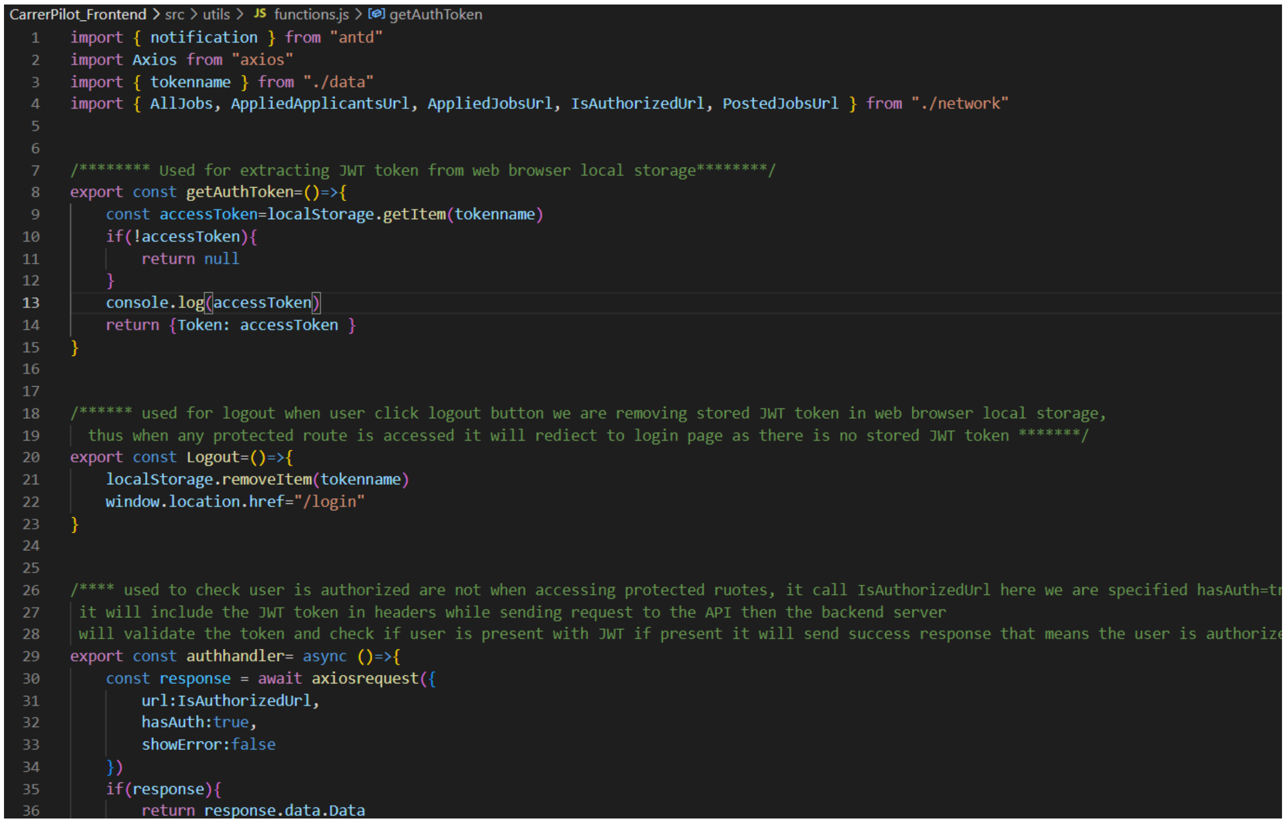
Table: golang_stud

Columns:

id	int AI PK
username	varchar(PK
password	varchar(
first_name	varchar(
last_name	varchar(
email	varchar(
phone_number	varchar(
role	varchar(

Overview of Code:

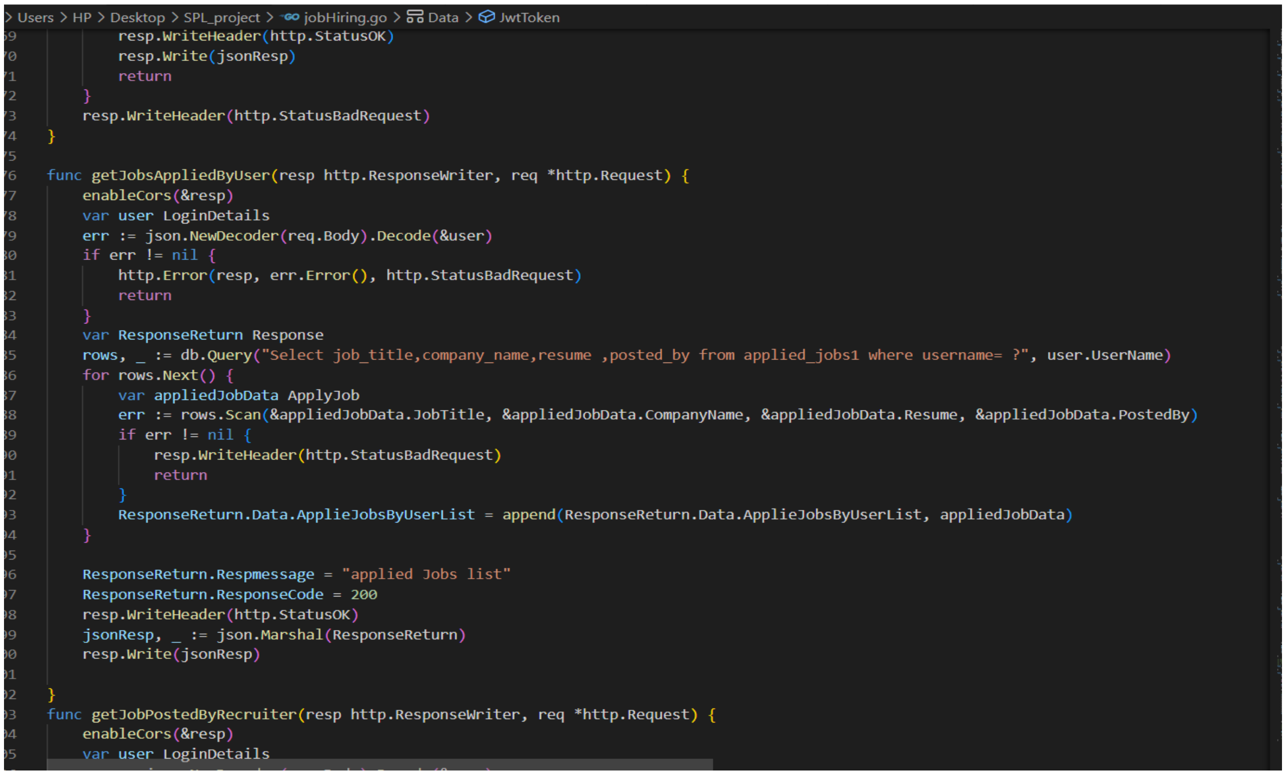
Front end JS code overview:



CarrerPilot_Frontend > src > utils > JS functions.js > getAuthToken

```js
import { notification } from "antd"
import Axios from "axios"
import { tokenname } from "./data"
import { AllJobs, AppliedApplicantsUrl, AppliedJobsUrl, IsAuthorizedUrl, PostedJobsUrl } from "./network"


/******** Used for extracting JWT token from web browser local storage********/
export const getAuthToken=()=>{
    const accessToken=localStorage.getItem(tokenname)
    if(!accessToken){
        return null
    }
    console.log(accessToken)
    return {Token: accessToken }
}


/****** used for logout when user click logout button we are removing stored JWT token in web browser local storage,
  thus when any protected route is accessed it will rediect to login page as there is no stored JWT token *******/
export const Logout=()=>{
    localStorage.removeItem(tokenname)
    window.location.href="/login"
}


/**** used to check user is authorized are not when accessing protected ruotes, it call IsAuthorizedUrl here we are specified hasAuth=tr
 it will include the JWT token in headers while sending request to the API then the backend server
 will validate the token and check if user is present with JWT if present it will send success response that means the user is authorize
export const authhandler= async ()=>{
    const response = await axiosrequest({
        url:IsAuthorizedUrl,
        hasAuth:true,
        showError:false
    })
    if(response){
        return response.data.Data
```

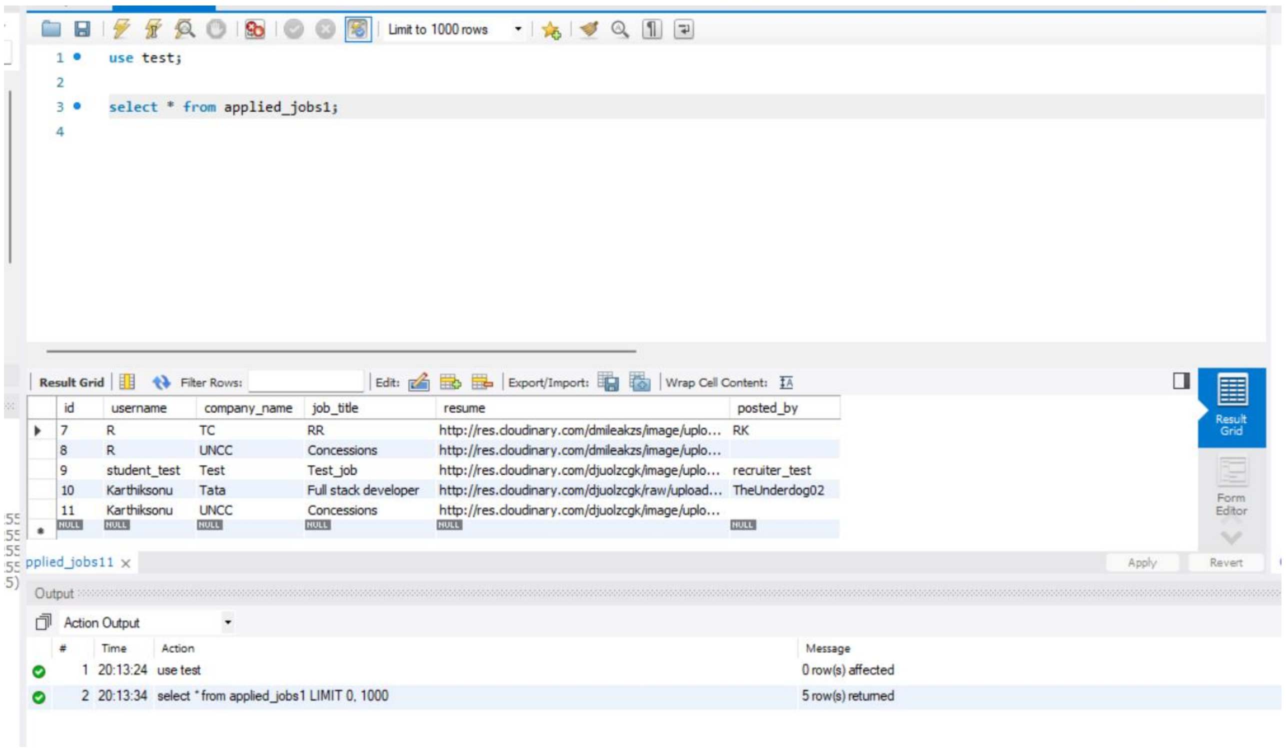
Go Lang code overview:



> Users > HP > Desktop > SPL_project > jobHiring.go > Data > JwtToken

```go
        resp.WriteHeader(http.StatusOK)
        resp.Write(jsonResp)
        return
    }
    resp.WriteHeader(http.StatusBadRequest)
}

func getJobsAppliedByUser(resp http.ResponseWriter, req *http.Request) {
    enableCors(&resp)
    var user LoginDetails
    err := json.NewDecoder(req.Body).Decode(&user)
    if err != nil {
        http.Error(resp, err.Error(), http.StatusBadRequest)
        return
    }
    var ResponseReturn Response
    rows, _ := db.Query("Select job_title,company_name,resume ,posted_by from applied_jobs1 where username= ?", user.UserName)
    for rows.Next() {
        var appliedJobData ApplyJob
        err := rows.Scan(&appliedJobData.JobTitle, &appliedJobData.CompanyName, &appliedJobData.Resume, &appliedJobData.PostedBy)
        if err != nil {
            resp.WriteHeader(http.StatusBadRequest)
            return
        }
        ResponseReturn.Data.AppliedJobsByUserList = append(ResponseReturn.Data.AppliedJobsByUserList, appliedJobData)
    }

    ResponseReturn.Respmessage = "applied Jobs list"
    ResponseReturn.ResponseCode = 200
    resp.WriteHeader(http.StatusOK)
    jsonResp, _ := json.Marshal(ResponseReturn)
    resp.Write(jsonResp)

}
func getJobPostedByRecruiter(resp http.ResponseWriter, req *http.Request) {
    enableCors(&resp)
    var user LoginDetails
```

Database table:



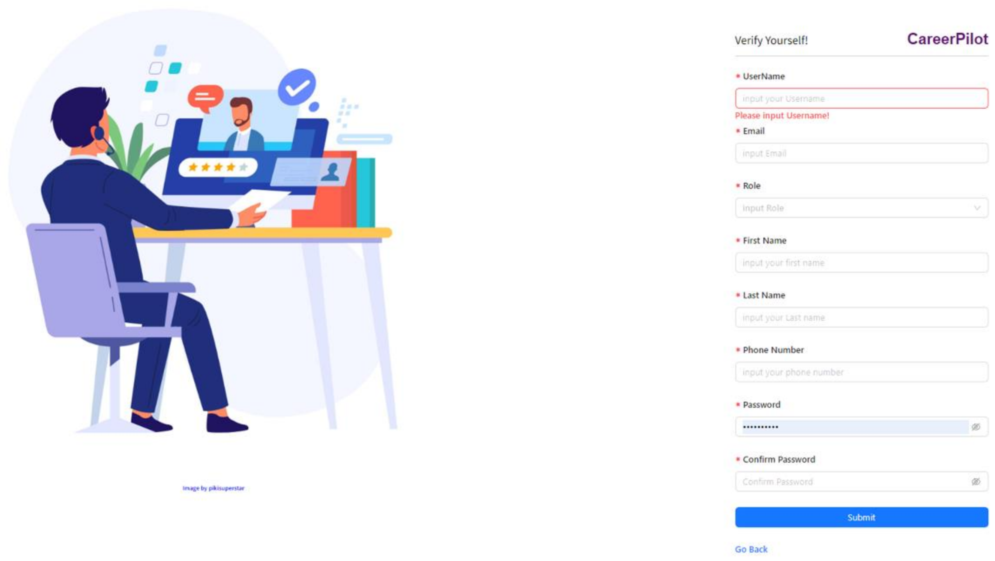
```
use test;

select * from applied_jobs1;
```

id	username	company_name	job_title	resume	posted_by
7	R	TC	RR	http://res.cloudinary.com/dmileakzs/image/uplo...	RK
8	R	UNCC	Concessions	http://res.cloudinary.com/dmileakzs/image/uplo...	
9	student_test	Test	Test_job	http://res.cloudinary.com/djuolzcgk/image/uplo...	recruiter_test
10	Karthiksonu	Tata	Full stack developer	http://res.cloudinary.com/djuolzcgk/raw/upload...	TheUnderdog02
11	Karthiksonu	UNCC	Concessions	http://res.cloudinary.com/djuolzcgk/image/uplo...	
NULL	NULL	NULL	NULL	NULL	NULL

#	Time	Action	Message
1	20:13:24	use test	0 row(s) affected
2	20:13:34	select * from applied_jobs1 LIMIT 0, 1000	5 row(s) returned

Screenshots for Sample Run in the Program:

Sign up page



Verify Yourself! CareerPilot

* UserName

input your Username

Please input Username!

* Email

input Email

* Role

input Role

* First Name

input your first name

* Last Name

input your Last name

* Phone Number

input your phone number

* Password

••••••••••

* Confirm Password

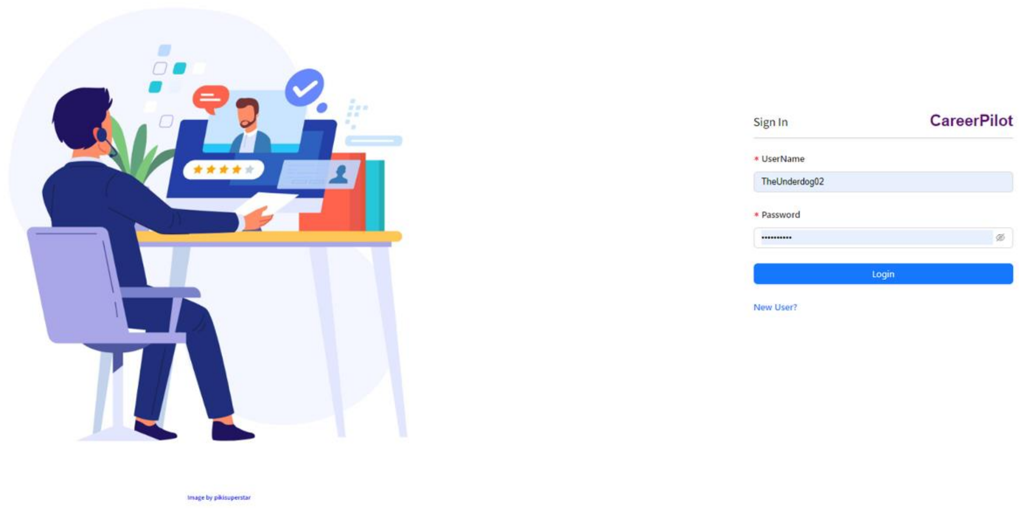
Confirm Password

Submit

Go Back

Any candidate be it’s a recruiter or a student, they initially needs to register to sign up the portal first the mandatary fields such as Username, Email, Role, First Name, Last Name, Phone Number, Password and Confirm Password

Sign in page



Sign In CareerPilot

* UserName

TheUnderdog02

* Password

••••••••••

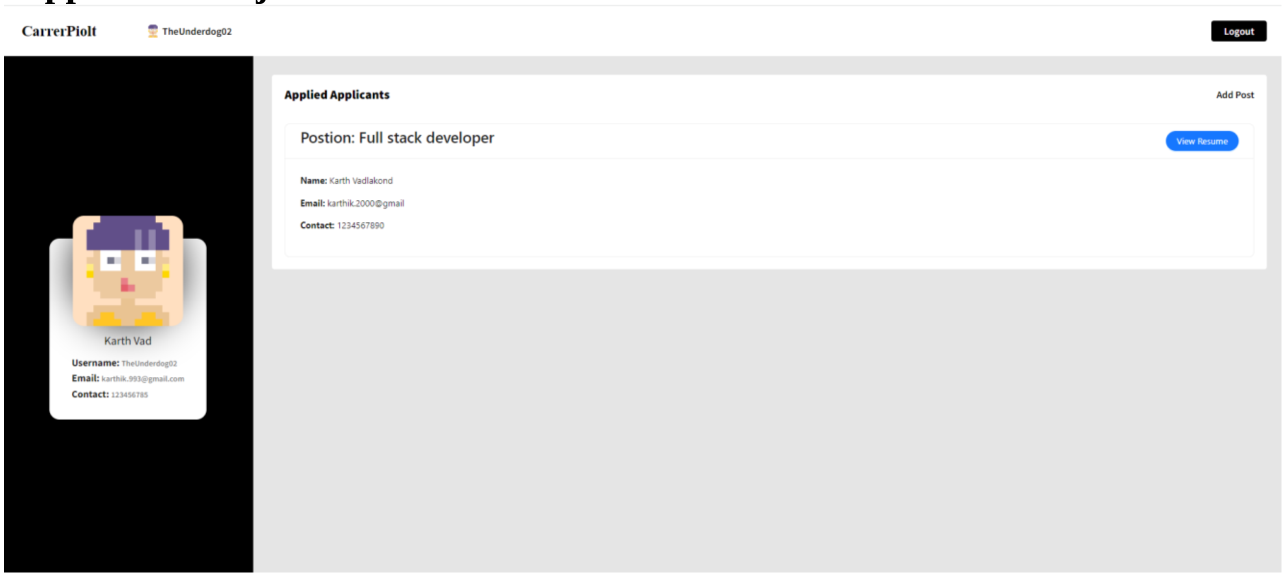
Login

New User?

A candidate need to give their credentials to sign in to the portal

Recruiter Portal

Applicants for job:



Applied Applicants

Postion: Full stack developer

Name: Karth Vadlakond

Email: karthik.2000@gmail

Contact: 1234567890

Karth Vad

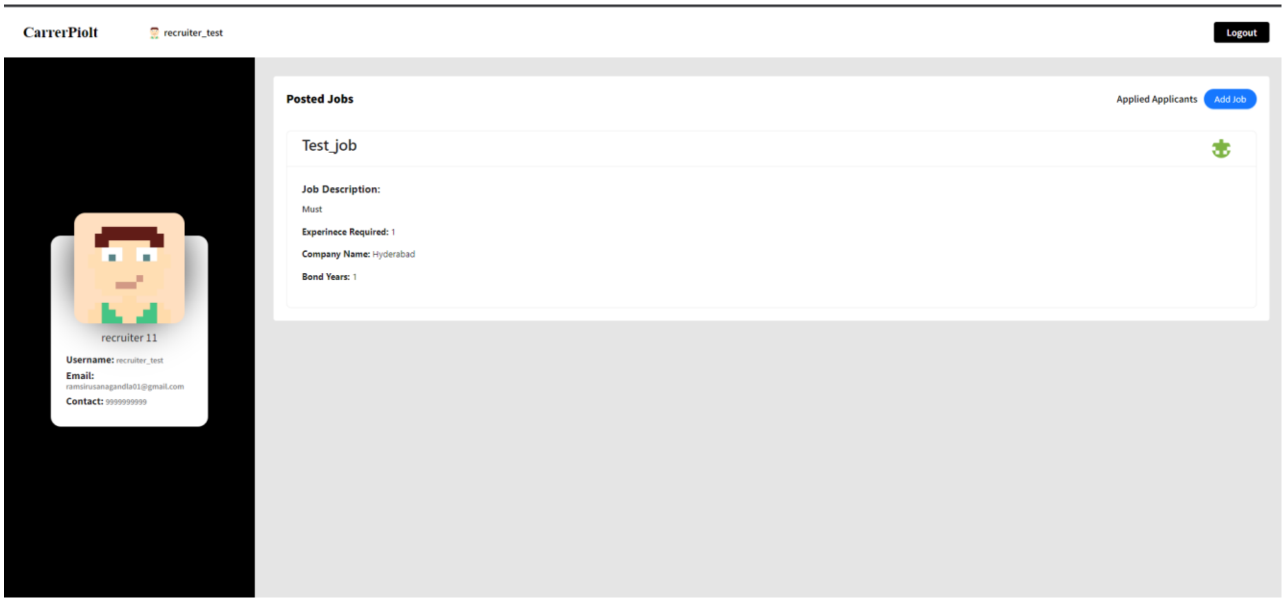
Username: TheUnderdog02

Email: karthik.993@gmail.com

Contact: 123456785

Recruiter can view the applied applicants and also their resumes which they approved

Posted Job Portal:



Posted Jobs

Test_job

Job Description:

Must

Experinece Required: 1

Company Name: Hyderabad

Bond Years: 1

recruiter 11

Username: recruiter_test

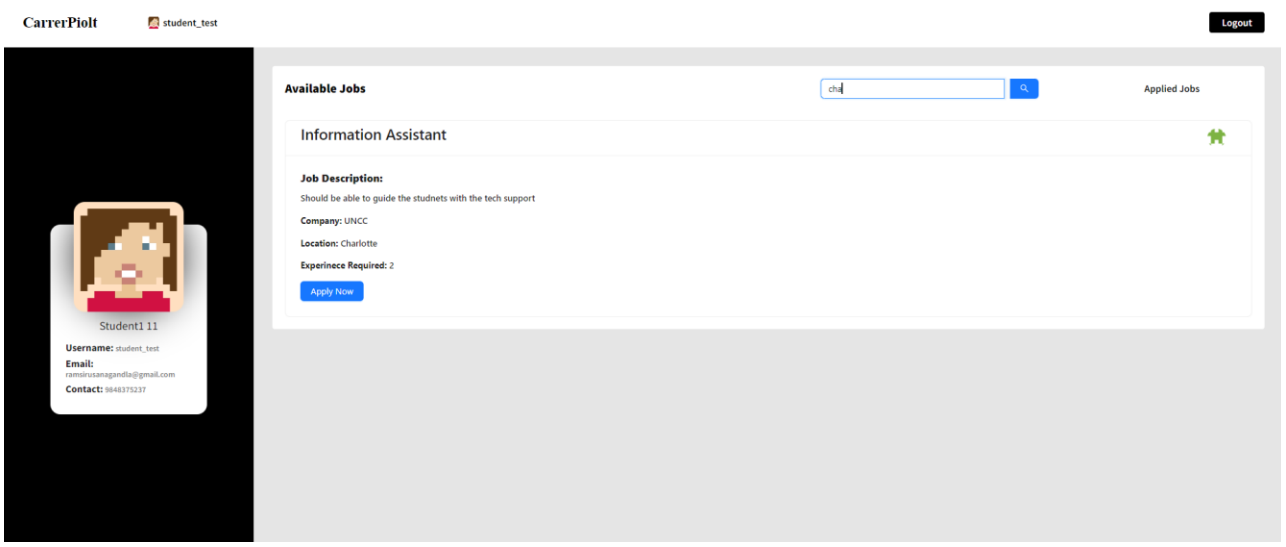
Email: ramsirusanagandla01@gmail.com

Contact: 9999999999

Recruiter can view the jobs that he posted and also can add new job postings

Student Portal

Search jobs:


CarrerPiolt

student_test

Logout

Available Jobs

cha

Applied Jobs

Information Assistant

Job Description:

Should be able to guide the studnets with the tech support

Company: UNCC

Location: Charlotte

Experinece Required: 2

Apply Now

Student1 11

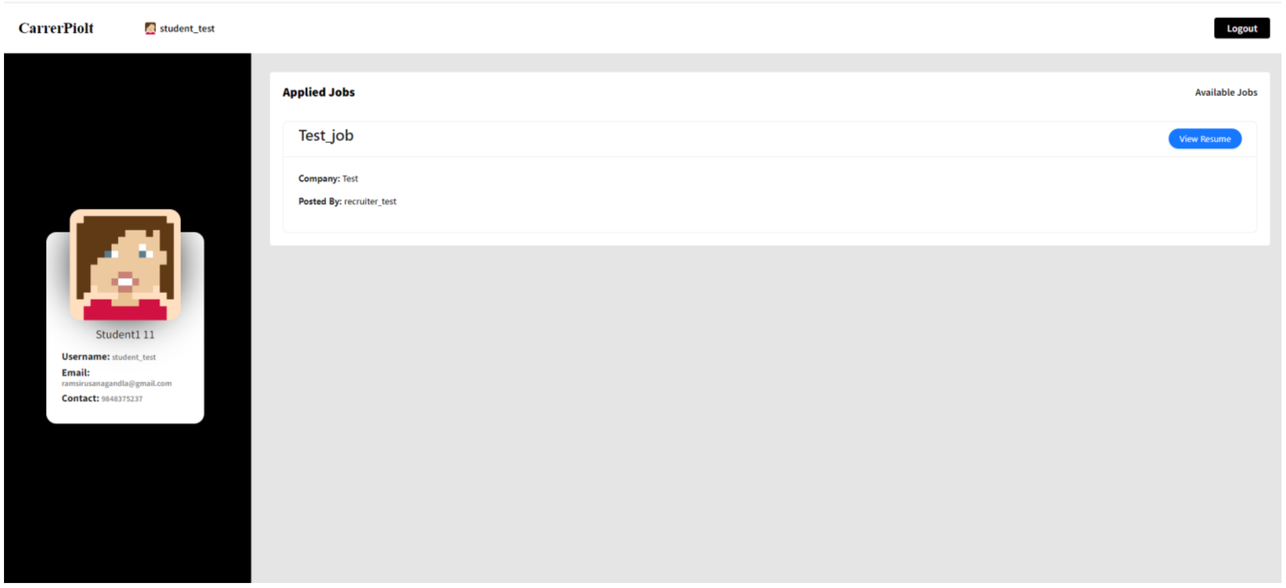
Username: student_test

Email: ramsirusanagandla@gmail.com

Contact: 9848375237

The search box makes it simple to navigate and filter job openings based on location, job, and company.

Applied Jobs


CarrerPiolt

student_test

Logout

Applied Jobs

Available Jobs

Test_job

View Resume

Company: Test

Posted By: recruiter_test

Student1 11

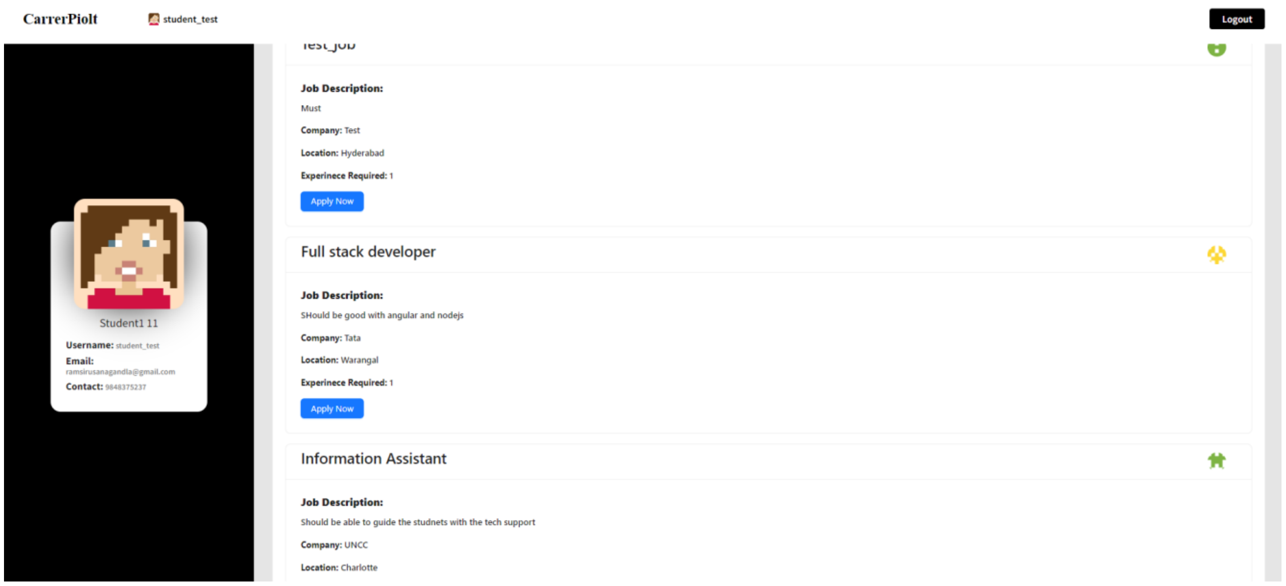
Username: student_test

Email: ramsirusanagandla@gmail.com

Contact: 9848375237

Students can view the jobs that he applied and also can apply new jobs.

Available jobs:



CarrerPiolt

student_test

Logout

Student1 11

Username: student_test

Email: ramsirusanagandla@gmail.com

Contact: 9848375237

Job Description: Must

Company: Test

Location: Hyderabad

Experinece Required: 1

Apply Now

Full stack developer

Job Description: SHould be good with angular and nodejs

Company: Tata

Location: Warangal

Experinece Required: 1

Apply Now

Information Assistant

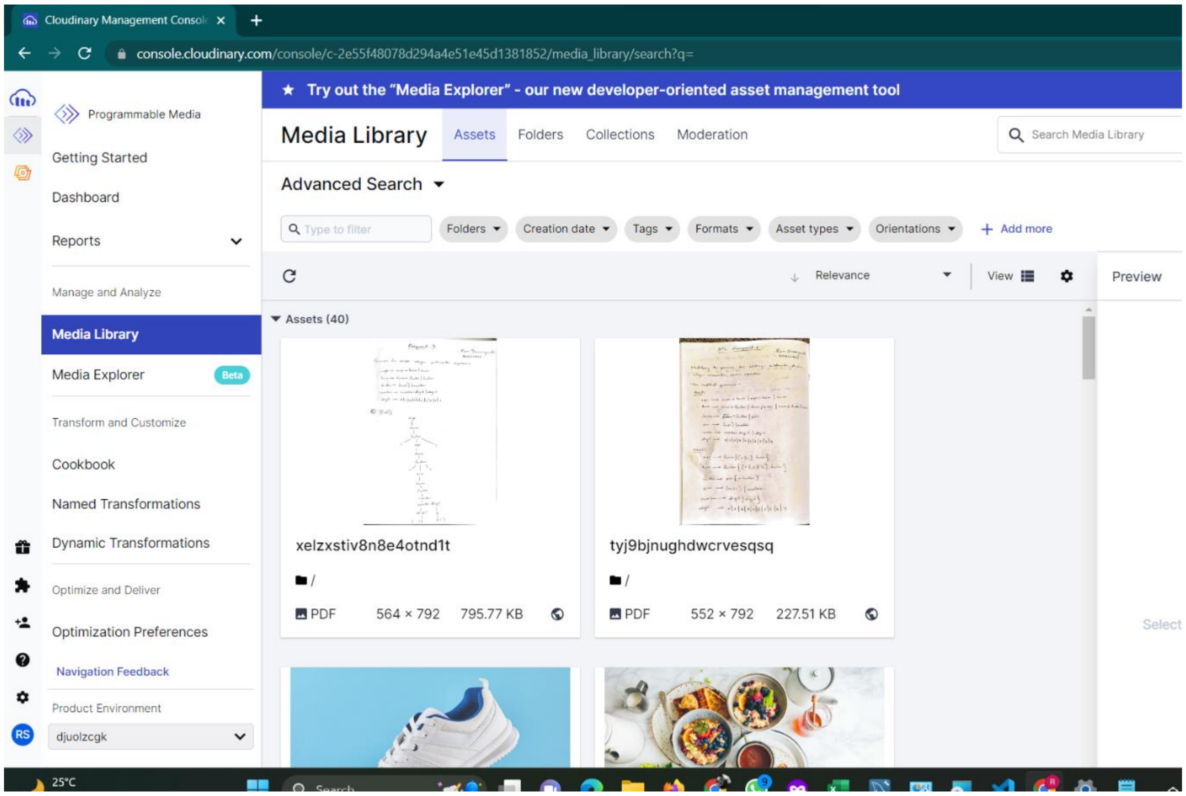
Job Description: Should be able to guide the studnets with the tech support

Company: UNCC

Location: Charlotte

Students can see job openings posted by recruiters from different companies

Cloudinary:



xelzxstiv8n8e4otnd1t	PDF	564 × 792	795.77 KB
tyj9bjnughdwcrvesqsq	PDF	552 × 792	227.51 KB

Cloudinary is a free platform and been used to store the resumes of the students and this stored resume link will be given to recruiter. Resumes stored in cloudinary will only be visible to the owner of that account but not by recruiters or student.

Test-Bed Description:

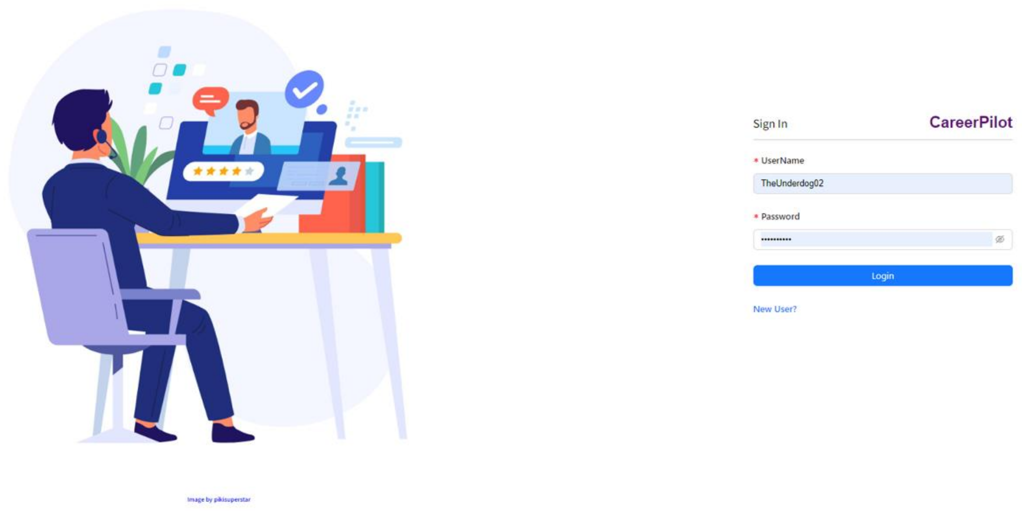
Hardware Configurations:
Windows 7 and higher
64-bit Intel or AMD CPU
4 GB RAM
4 GB free disk space

Software Configurations:
Visual Studio Code 2022 My
SQL Workbench 8.0 CE
Node js
ReactJS
Chrome/Internet Explorer/Mozilla/Edge Browsers

Observations:

We have tested our portal by signing up and logging in we even added few jobs and applied to few roles and see he they are reflected in respective portals.

Login page



Sign In CareerPilot

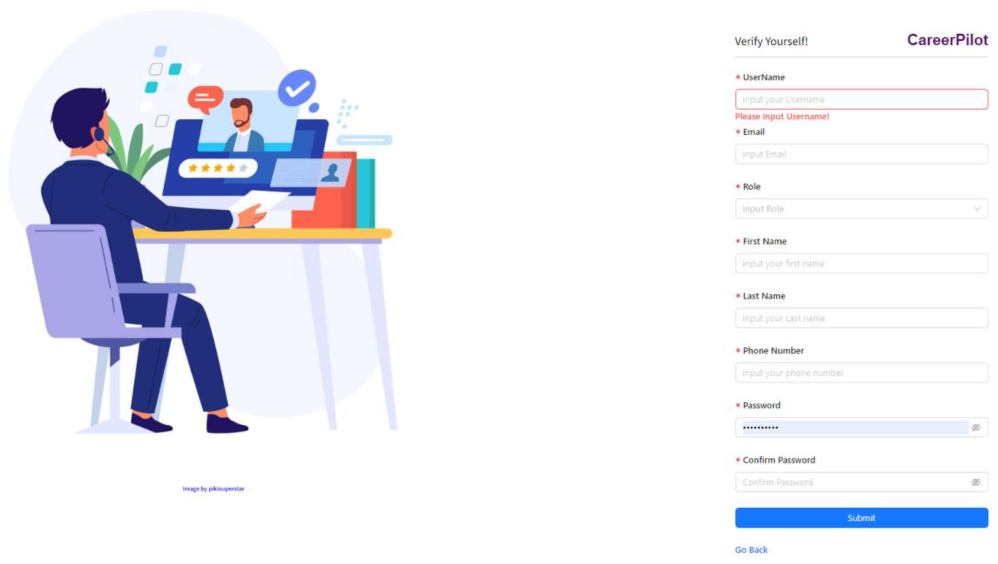
* UserName

TheUnderdog02

* Password

••••••••••

Sign up page



Verify Yourself! CareerPilot

* UserName

input your Username

Please input Username!

* Email

input Email

* Role

input Role

* First Name

input your first name

* Last Name

input your Last name

* Phone Number

input your phone number

* Password

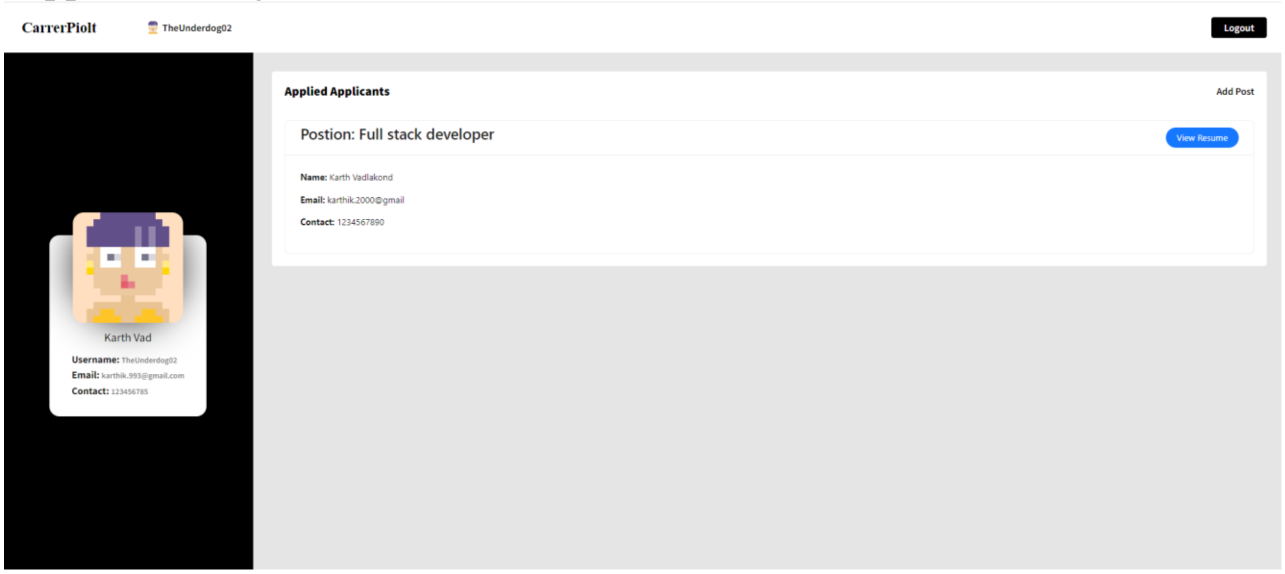
••••••••••

* Confirm Password

Confirm Password

Recruiter Portal

Applicants for job:


CarrerPiolt TheUnderdog02

Logout

Karth Vad

Username: TheUnderdog02
Email: karthik.993@gmail.com
Contact: 123456785

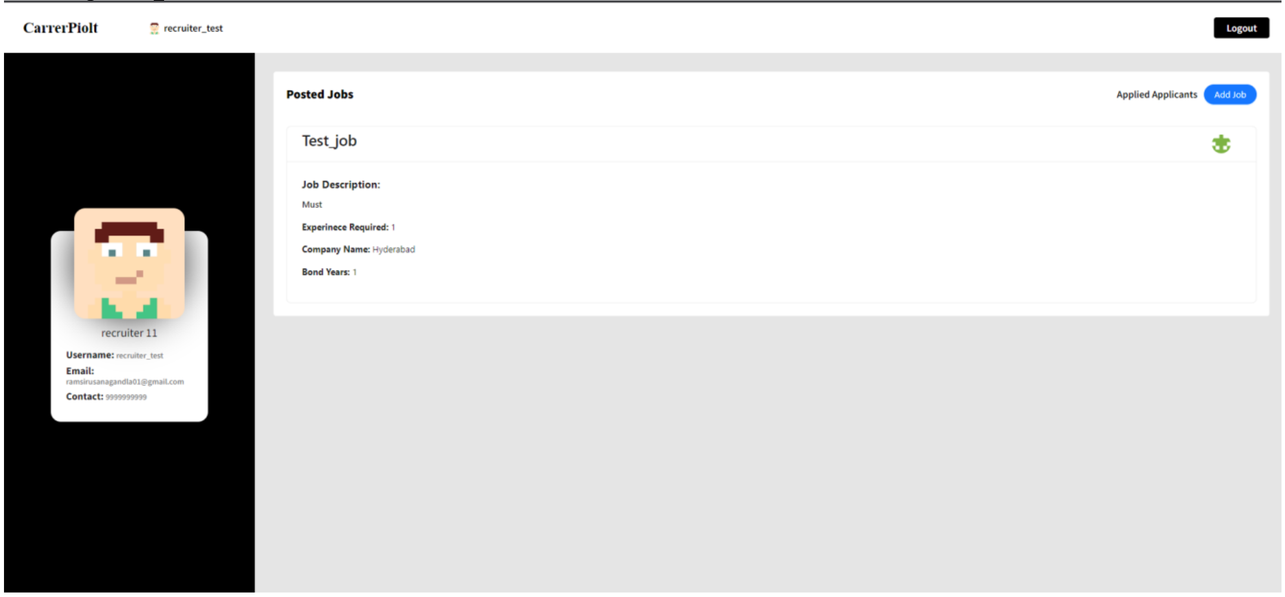
Applied Applicants Add Post

Postion: Full stack developer

View Resume

Name: Karth Vadlakond
Email: karthik.2000@gmail
Contact: 1234567890

Add job portal:


CarrerPiolt recruiter_test

Logout

recruiter 11

Username: recruiter_test
Email: ramsirusanagandla01@gmail.com
Contact: 9999999999

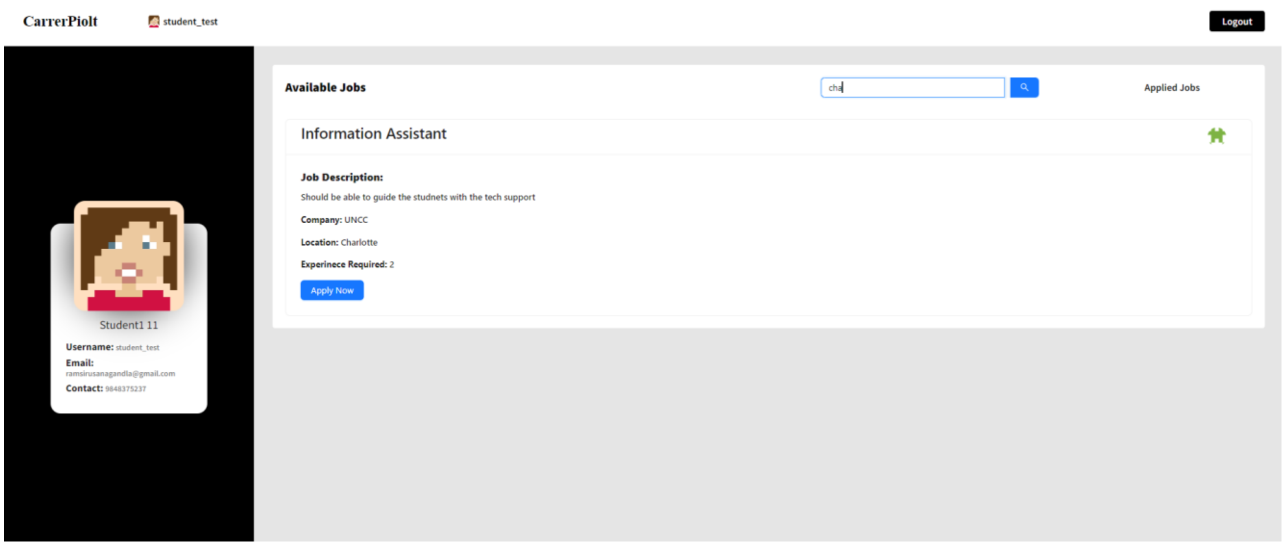
Posted Jobs Applied Applicants Add Job

Test_job

Job Description:
Must
Experinece Required: 1
Company Name: Hyderabad
Bond Years: 1

Student Portal

Search jobs:


CarrerPiolt

student_test

Logout

Student1 11

Username: student_test

Email: ramsirusanagandla@gmail.com

Contact: 9848375237

Available Jobs

cha

Applied Jobs

Information Assistant

Job Description:

Should be able to guide the studnets with the tech support

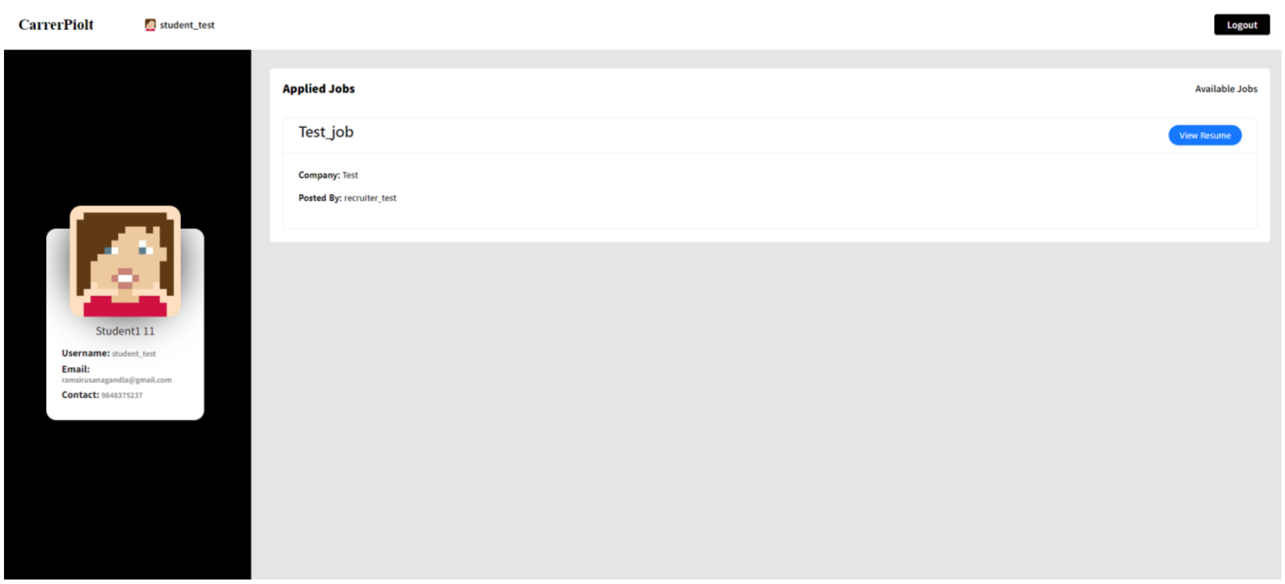
Company: UNCC

Location: Charlotte

Experinece Required: 2

Apply Now

Applied Jobs:


CarrerPiolt

student_test

Logout

Student1 11

Username: student_test

Email: ramsirusanagandla@gmail.com

Contact: 9848375237

Applied Jobs

Available Jobs

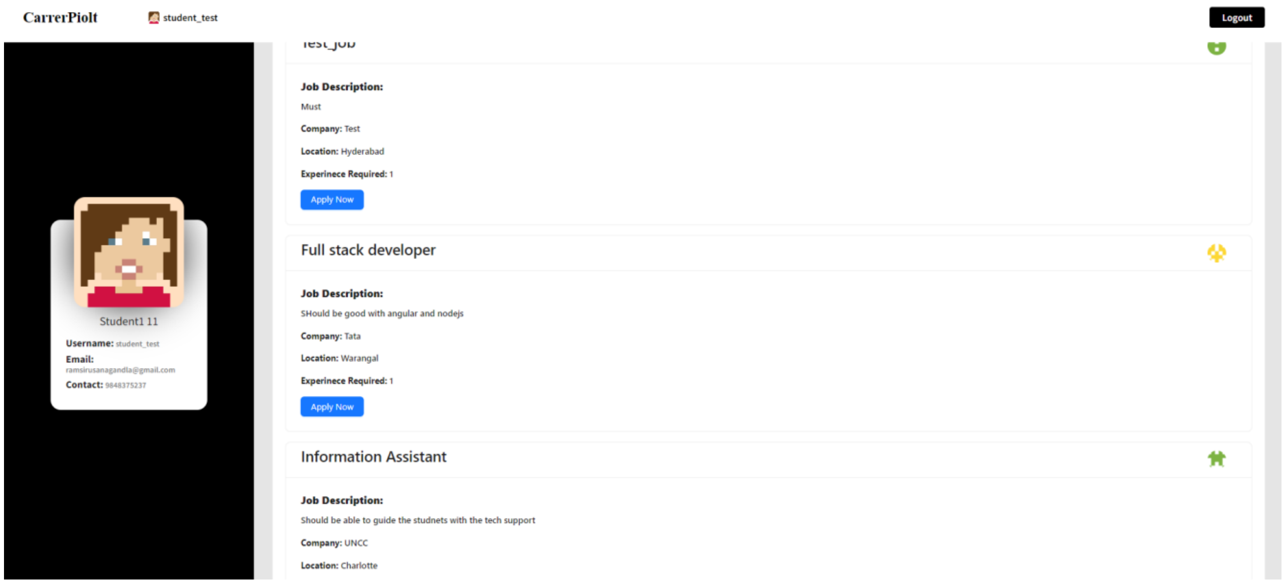
Test_job

View Resume

Company: Test

Posted By: recruiter_test

Available jobs:



CarrerPiolt · student_test · Logout

Student1 11

Username: student_test

Email: ramsirusanagandla@gmail.com

Contact: 9848375237

Test_job

Job Description:
Must

Company: Test

Location: Hyderabad

Experinece Required: 1

Apply Now

Full stack developer

Job Description:
SHould be good with angular and nodejs

Company: Tata

Location: Warangal

Experinece Required: 1

Apply Now

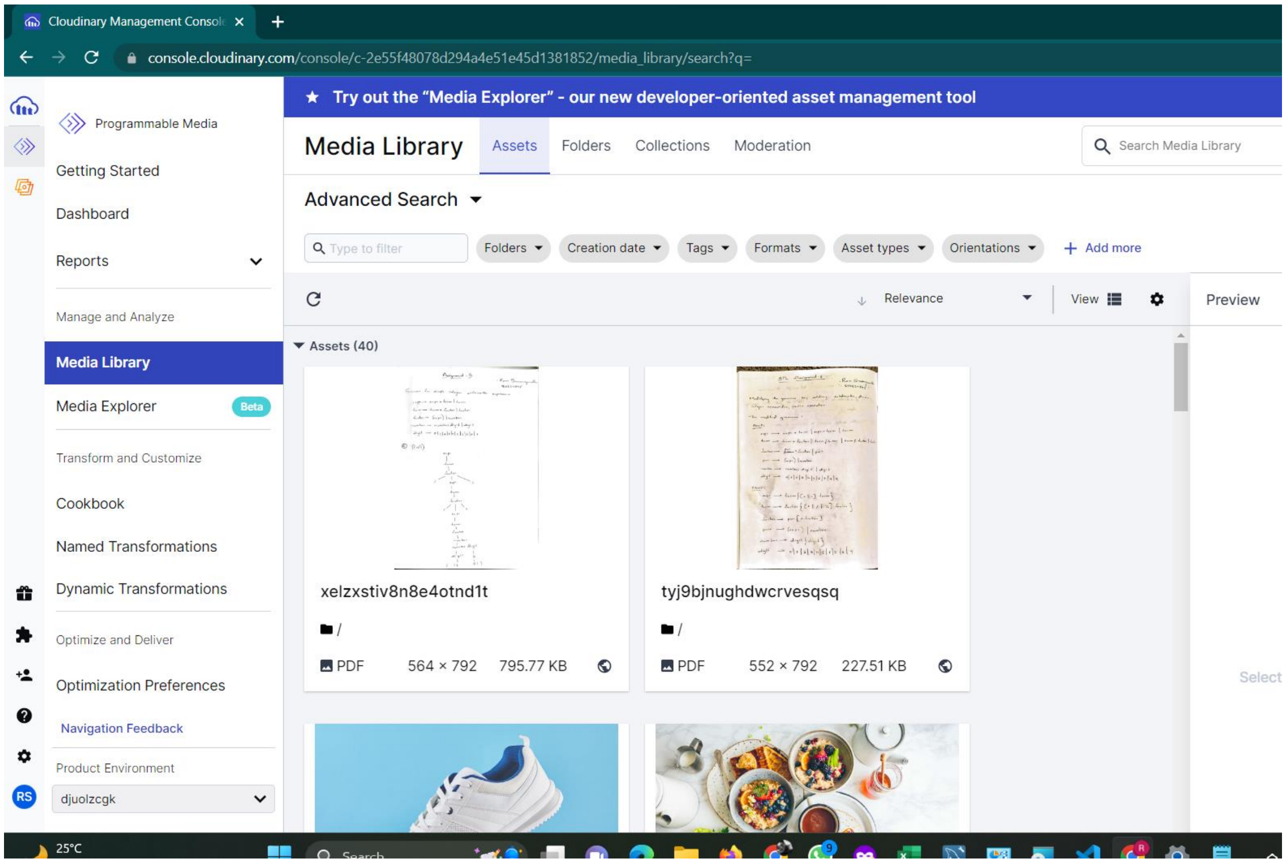
Information Assistant

Job Description:
Should be able to guide the studnets with the tech support

Company: UNCC

Location: Charlotte

Cloudinary:



Cloudinary Management Console

console.cloudinary.com/console/c-2e55f48078d294a4e51e45d1381852/media_library/search?q=

★ Try out the "Media Explorer" - our new developer-oriented asset management tool

Media Library · Assets · Folders · Collections · Moderation

Advanced Search

Folders · Creation date · Tags · Formats · Asset types · Orientations · Add more

Relevance · View · Preview

Assets (40)

Name	Folder	Format	Dimensions	Size
xelzxstiv8n8e4otnd1t	/	PDF	564 × 792	795.77 KB
tyj9bjnughdwcrvesqsq	/	PDF	552 × 792	227.51 KB

Future Scope:

- In the future, we can include chat bots to this portal so that the recruiter and applicant directly contact each other for updates or queries.
- The portal could be used to facilitate internships and co-op programs, providing students with valuable work experience while still in school.
- The portal could be expanded to include alumni, creating a more comprehensive network of job seekers and recruiters and offering new opportunities to alumni of the university.

Conclusion:

We were able to create a job portal by using Golang which will help both the recruiter and students by eliminating the gap between them.