****

**COURSE CODE: BCA-605**

**TITLE OF THE PROJECT**

**Submitted to the**

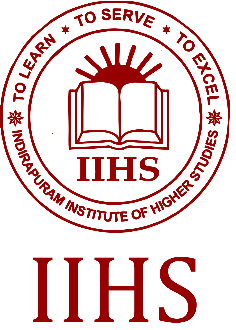
**in partial fulfilment of the requirements**

**for the award of the degree**

**Bachelor of Computer Application (BCA)**

**Year of Submission**

|  |  |
| --- | --- |
| **SUPERVISOR’S NAME:- PRERNA BHATNAGAR** | **STUDENT’S NAME:-** |
| **& DESIGNATION:-** | **ENROLMENT NO.:-** |
|  | **ROLL NO.:- 223971060** |



**INDIRAPURAM INSTITUTE OF HIGHER STUDIES**

**IIHS-Vishisht Shiksha Bhookhand, Nyay Khand -1 Indirapuram,Ghaziabad (UP)-201014**

**+91-120-2607101,9560994644 | Email:** [**contact@theiihs.com**](mailto:contact@theiihs.com) **| Website:** [**www.theiihs.com**](http://www.theiihs.com)

**PODIUM**

**- AI-Powered Resume Management System Comprehensive Project Report**

**TABLE OF CONTENTS**

**1. Introduction and Objectives**

**2. Project Category and Technology Stack**

**3. System Requirements**

**4. Problem Definition and Requirements**

**5. System Analysis**

**6. System Design**

**7. Implementation**

**8. Testing**

**9. Security Measures**

**10. Cost Estimation**

**11. Future Scope and Enhancements**

**12. Bibliography**

**13. Appendices**

**14. Glossary**

**1. INTRODUCTION AND OBJECTIVES**

=============================

**1.1 Executive Summary**

--------------------

Podium is a modern resume management system that helps companies streamline their hiring process. By combining artificial intelligence with user-friendly design, we've created a platform that makes resume screening faster, fairer, and more efficient.

The system helps companies by:

- Automating the initial resume screening process

- Reducing bias in candidate evaluation

- Providing clear insights about candidates

- Making team collaboration easier

- Keeping candidate data secure

- Offering an intuitive interface for all users

**1.2 Project Overview**

-------------------

Podium brings together several key technologies to create a powerful hiring solution:

1. Smart Resume Processing:

- Automatically extracts information from resumes

- Identifies key skills and experience

- Organizes candidate data efficiently

- Validates information automatically

2. Modern Web Platform:

- Fast and responsive interface

- Works on all devices

- Easy to use and navigate

- Quick access to information

3. Security Features:

- Secure data storage

- Controlled access to information

- Activity tracking

- Privacy protection

4. Team Collaboration:

- Easy communication between team members

- Simple file sharing

- Clear feedback system

- Progress tracking

**1.3 Problem Statement**

--------------------

Companies face several key challenges in their hiring process that Podium aims to solve:

1. Data Handling Issues:

- Different Resume Formats

\* Various file types

\* Different layouts

\* Different writing styles

\* Multiple languages

- Manual Data Entry

\* Takes too much time

\* Prone to errors

\* Inconsistent format

\* Data checking issues

- Processing Problems

\* Human mistakes

\* Inconsistent data

\* Missing information

\* Duplicate entries

- Slow Screening

\* Manual review

\* Too many applications

\* Resource heavy

\* Slow responses

2. Evaluation Challenges:

- Human Bias

\* Personal preferences

\* Inconsistent standards

\* Different views

\* Unconscious bias

- Inconsistent Standards

\* Different criteria

\* Various methods

\* No clear rules

\* Unclear measures

- Limited Insights

\* Basic analysis

\* Missing connections

\* Incomplete review

\* Limited assessment

- Poor Matching

\* Basic keyword search

\* Limited understanding

\* Wrong matches

\* Poor fit

3. Team Work Issues:

- Poor Communication

\* Isolated decisions

\* Slow feedback

\* Poor sharing

\* Communication gaps

- Slow Feedback

\* Delayed responses

\* Incomplete feedback

\* Lost messages

\* Slow decisions

- File Sharing Problems

\* Version issues

\* Access problems

\* Sharing barriers

\* Security worries

- Update Issues

\* Slow changes

\* Missing alerts

\* Wrong information

\* Poor sync

**1.4 Objectives**

-------------

Our main goals for Podium are:

1. Technical Goals:

- Create accurate resume parsing

- Build a reliable system

- Ensure data security

- Make the interface user-friendly

- Maintain system performance

- Connect with other tools

2. Functional Goals:

- Automate resume screening

- Match candidates to jobs

- Enable team collaboration

- Generate useful reports

- Support all resume formats

- Provide advanced search

3. Business Goals:

- Speed up hiring

- Improve candidate evaluation

- Increase team productivity

- Reduce costs

- Improve candidate experience

- Better hiring decisions

**1.5 Target Users**

---------------

The system is designed for:

1. Primary Users:

- HR Teams

\* Recruiters

\* HR managers

\* HR administrators

\* Hiring managers

- Recruitment Teams

\* Recruiters

\* Sources1

\* Coordinators

\* Team leads

- Hiring Managers

\* Department heads

\* Team managers

\* Project leaders

\* Technical leads

- Talent Acquisition Teams

\* Recruiters

\* Consultants

\* Talent scouts

\* Career advisors

2. Secondary Users:

- Career Counselors

\* Career advisors

\* Job coaches

\* Placement officers

\* Career specialists

- Job Seekers

\* Active candidates

\* Passive candidates

\* Career changers

\* Recent graduates

- Department Heads

\* Business leaders

\* Division managers

\* Project directors

\* Executive team

- IT Support

\* System administrators

\* Support staff

\* Security officers

\* Database managers

**1.6 Key Features**

---------------

The system offers these main features:

1. Resume Management:

- Multiple file formats

\* PDF support

\* Word documents

\* Text files

\* Image-based resumes

- Automatic data extraction

\* Contact details

\* Education history

\* Work experience

\* Skills list

- Version tracking

\* Document history

\* Change tracking

\* Update alerts

\* Version comparison

- Template system

\* Custom templates

\* Template library

\* Format options

\* Style choices

2. Smart Analysis:

- Skills identification

\* Technical skills

\* Soft skills

\* Skill levels

\* Skill gaps

- Experience evaluation

\* Job roles

\* Industry experience

\* Project work

\* Leadership roles

- Education verification

\* Degree checking

\* Institution validation

\* Course analysis

\* Academic performance

- Candidate scoring

\* Overall rating

\* Role matching

\* Skill assessment

\* Experience evaluation

3. Team Tools:

- Chat system

\* Team messaging

\* Candidate communication

\* File sharing

\* Status updates

- File sharing

\* Document sharing

\* Version control

\* Access management

\* Download tracking

- Feedback system

\* Comments

\* Feedback forms

\* Ratings

\* Discussion boards

- Status tracking

\* Application status

\* Interview stages

\* Evaluation progress

\* Decision tracking

4. Security Features:

- Access control

\* User roles

\* Permission levels

\* Access limits

\* Activity tracking

- Data protection

\* Secure storage

\* Safe transmission

\* Data backup

\* Key management

- Activity logging

\* User actions

\* System events

\* Security alerts

\* Compliance reports

- Secure storage

\* Encrypted files

\* Access control

\* Backup system

\* Recovery options

**2. PROJECT CATEGORY AND TECHNOLOGY STACK**

=======================================

**2.1 Project Category**

-------------------

Podium combines several key areas of technology to create a complete hiring solution:

1. Artificial Intelligence

- Text understanding

\* Reading and analyzing resumes

\* Finding important information

\* Understanding context

\* Learning from examples

- Smart matching

\* Finding patterns

\* Making predictions

\* Sorting information

\* Learning from data

- Deep learning

\* Understanding complex patterns

\* Finding hidden connections

\* Improving over time

\* Making smart decisions

2. Supporting Areas:

- Web Development

\* User interface

\* Backend services

\* System connections

\* Design

- Data Management

\* Information storage

\* Quick searching

\* Data organization

\* Analysis

- Cloud Services

\* Online storage

\* Remote access

\* System hosting

\* Automatic scaling

- User Interface

\* Easy navigation

\* Clear design

\* Mobile support

\* Accessibility

**2.2 Technology Stack**

**-**------------------

We use modern, reliable tools to build Podium:

1. Frontend Tools:

- Next.js 14

\* Fast page loading

\* Easy navigation

\* Quick updates

\* Simple routing

- TypeScript

\* Better code quality

\* Fewer errors

\* Clear structure

\* Easy maintenance

- Tailwind CSS

\* Quick styling

\* Mobile-friendly

\* Custom designs

\* Easy changes

- React

\* Interactive pages

\* Quick updates

\* Reusable parts

\* Smooth experience

- Shadcn UI

\* Ready-made parts

\* Consistent look

\* Easy access

\* Custom options

- Redux Toolkit

\* Data management

\* State control

\* Easy updates

\* Better performance

2. Backend Tools:

- Django 4.2

\* Solid foundation

\* Admin tools

\* Form handling

\* Easy setup

- Django REST Framework

\* API building

\* Data handling

\* User access

\* Security

- Python 3.10+

\* Easy coding

\* Fast processing

\* Good libraries

\* Error handling

- PostgreSQL

\* Reliable storage

\* Quick searching

\* Data safety

\* Easy backup

- Redis

\* Fast caching

\* Session handling

\* Message passing

\* Real-time features

- Celery

\* Background tasks

\* Scheduled jobs

\* Task management

\* Better performance

3. AI/ML Tools:

- TensorFlow

\* Smart learning

\* Pattern finding

\* Model training

\* Quick results

- PyTorch

\* Deep learning

\* Research tools

\* Model building

\* Custom solutions

- NLTK

\* Text processing

\* Word analysis

\* Language understanding

\* Information extraction

- SpaCy

\* Language processing

\* Information finding

\* Text analysis

\* Quick results

- Scikit-learn

\* Machine learning

\* Data preparation

\* Model testing

\* Feature selection

4. Cloud Services:

- AWS S3

\* File storage

\* Easy access

\* Safe backup

\* Fast delivery

- Vercel

\* Website hosting

\* Quick updates

\* Easy setup

\* Performance tracking

- Cloudflare

\* Fast delivery

\* Attack protection

\* Secure access

\* Easy management

- Docker

\* Easy deployment

\* Consistent setup

\* Simple updates

\* Better testing

- Kubernetes

\* System management

\* Easy scaling

\* Load balancing

\* Service discovery

**2.3 Development Tools**

**--------------------**

We use these tools to build and maintain Podium:

1. Version Control:

- Git

\* Code tracking

\* Change history

\* Team work

\* Easy fixes

- GitHub

\* Code storage

\* Team review

\* Issue tracking

\* Automatic tasks

- GitLab CI/CD

\* Automatic testing

\* Easy deployment

\* Task automation

\* Environment setup

2. Development Environment:

- VS Code

\* Code editing

\* Easy debugging

\* Helpful tools

\* Quick access

- PyCharm

\* Python coding

\* Code checking

\* Testing tools

\* Database access

- Docker Desktop

\* Container setup

\* Easy testing

\* Network setup

\* Storage management

- Postman

\* API testing

\* Request checking

\* Environment setup

\* Automatic tests

3. Testing Tools:

- Jest

\* Code testing

\* Integration checks

\* Mock testing

\* Coverage reports

- PyTest

\* Python testing

\* Test setup

\* Multiple tests

\* Plugin support

- Selenium

\* Browser testing

\* UI checks

\* Cross-browser tests

\* Performance tests

- Cypress

\* End-to-end tests

\* Live testing

\* Easy debugging

\* Time tracking

**3. SYSTEM REQUIREMENTS**

=====================

**3.1 Hardware Requirements**

------------------------

1. Server Needs:

- CPU: 4+ cores

\* Modern processor

\* Good speed

\* Multi-tasking support

\* Virtual machine support

- RAM: 8GB minimum

\* Fast memory

\* Error checking

\* Expandable memory

\* Good performance

- Storage: 100GB SSD

\* Fast storage

\* Backup support

\* Easy replacement

\* Safe storage

- Network: 1Gbps

\* Fast connection

\* Backup connection

\* Load sharing

\* Security support

2. User Computer Needs:

- Modern Web Browser

\* Chrome 90+

\* Firefox 88+

\* Safari 14+

\* Edge 90+

- 4GB RAM

\* Basic memory

\* Memory management

\* Cache support

\* Good performance

- 2GB Free Storage

\* Local storage

\* Cache space

\* Temp files

\* Offline data

- Internet Connection

\* Good speed

\* Stable connection

\* Low delay

\* Backup option

**3.2 Software Requirements**

------------------------

1. Operating System:

- Windows 10/11

\* 64-bit system

\* Latest updates

\* Security updates

\* System needs

- macOS 10.15+

\* Intel/Apple chip

\* Latest updates

\* Security features

\* System needs

- Linux Ubuntu 20.04+

\* Long-term support

\* Server version

\* Security updates

\* System needs

2. Development Setup:

- Node.js 18+

\* Long-term support

\* Package manager

\* Development tools

\* Testing tools

- Python 3.10+

\* Virtual environment

\* Package manager

\* Development tools

\* Testing tools

- Git 2.30+

\* Version control

\* Branch support

\* Merge tools

\* Easy fixes

- Docker 20.10+

\* Container support

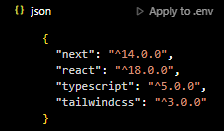
\* Image management

\* Network setup

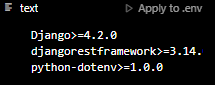
\* Storage setup

3. Required Software:

Frontend:



Backend:



**3.3 Network Requirements**

**-----------------------**

1. Server Network:

- Speed

\* Fast connection

\* Easy upgrade

\* Load sharing

\* Traffic control

- Security

\* Firewall

\* Secure connection

\* Attack protection

\* Safe access

- Monitoring

\* Network checks

\* Performance tracking

\* Alert system

\* Activity logs

2. User Network:

- Connection

\* Good speed

\* Low delay

\* Stable connection

\* Backup option

- Security

\* Secure connection

\* Safe access

\* Firewall

\* Virus protection

- Performance

\* Low delay

\* Few lost packets

\* Stable connection

\* Speed control

**3.4 Security Requirements**

**------------------------**

1. User Access:

- Login Security

\* Two-factor login

\* OAuth support

\* Secure tokens

\* Session control

- Access Control

\* User roles

\* Permission setup

\* Resource protection

\* Activity logs

2. Data Protection:

- Security

\* Safe storage

\* Secure transfer

\* Key management

\* Certificate handling

- Backup

\* Regular backup

\* Recovery plan

\* Data keeping

\* Archive system

3. Standards:

- Rules

\* GDPR

\* HIPAA

\* SOC 2

\* ISO 27001

- Checks

\* Security review

\* Rule checking

\* Risk check

\* Problem scanning

**4. PROBLEM DEFINITION AND REQUIREMENTS**

**=====================================**

**4.1 Problem Definition**

**---------------------**

Companies face several key challenges in their hiring process that Podium aims to solve:

1. Data Handling Issues:

- Different Resume Formats

\* Various file types

\* Different layouts

\* Different writing styles

\* Multiple languages

- Manual Data Entry

\* Takes too much time

\* Prone to errors

\* Inconsistent format

\* Data checking issues

- Processing Problems

\* Human mistakes

\* Inconsistent data

\* Missing information

\* Duplicate entries

- Slow Screening

\* Manual review

\* Too many applications

\* Resource heavy

\* Slow responses

2. Evaluation Challenges:

- Human Bias

\* Personal preferences

\* Inconsistent standards

\* Different views

\* Unconscious bias

- Inconsistent Standards

\* Different criteria

\* Various methods

\* No clear rules

\* Unclear measures

- Limited Insights

\* Basic analysis

\* Missing connections

\* Incomplete review

\* Limited assessment

- Poor Matching

\* Basic keyword search

\* Limited understanding

\* Wrong matches

\* Poor fit

3. Team Work Issues:

- Poor Communication

\* Isolated decisions

\* Slow feedback

\* Poor sharing

\* Communication gaps

- Slow Feedback

\* Delayed responses

\* Incomplete feedback

\* Lost messages

\* Slow decisions

- File Sharing Problems

\* Version issues

\* Access problems

\* Sharing barriers

\* Security worries

- Update Issues

\* Slow changes

\* Missing alerts

\* Wrong information

\* Poor sync

**4.2 Functional Requirements**

**--------------------------**

1. User Management:

```typescript

interface User {

id: string;

email: string;

firstName: string;

lastName: string;

role: UserRole;

permissions: Permission[];

profile: UserProfile;

settings: UserSettings;

activity: UserActivity[];

}

enum UserRole {

ADMIN = 'ADMIN',

HR = 'HR',

MANAGER = 'MANAGER',

USER = 'USER',

GUEST = 'GUEST'

}

interface UserProfile {

avatar: string;

department: string;

position: string;

contact: ContactInfo;

preferences: UserPreferences;

}

interface UserSettings {

notifications: NotificationSettings;

privacy: PrivacySettings;

display: DisplaySettings;

language: LanguageSettings;

}

```

2. Resume Management:

```typescript

interface Resume {

id: string;

userId: string;

title: string;

content: string;

format: ResumeFormat;

status: ResumeStatus;

metadata: ResumeMetadata;

history: ResumeHistory[];

analysis: ResumeAnalysis;

createdAt: Date;

updatedAt: Date;

}

enum ResumeFormat {

PDF = 'PDF',

DOCX = 'DOCX',

TXT = 'TXT',

HTML = 'HTML',

RTF = 'RTF'

}

interface ResumeMetadata {

fileSize: number;

pageCount: number;

lastModified: Date;

checksum: string;

tags: string[];

}

interface ResumeHistory {

version: number;

changes: Change[];

timestamp: Date;

user: string;

}

```

3. Smart Analysis:

```typescript

interface Analysis {

id: string;

resumeId: string;

skills: Skill[];

experience: Experience[];

education: Education[];

score: number;

recommendations: string[];

insights: Insight[];

matches: JobMatch[];

}

interface Skill {

name: string;

level: SkillLevel;

years: number;

relevance: number;

category: SkillCategory;

}

interface Experience {

company: string;

position: string;

duration: Duration;

responsibilities: string[];

achievements: string[];

skills: string[];

}

interface Education {

institution: string;

degree: string;

field: string;

graduation: Date;

gpa: number;

achievements: string[];

}

```

**4.3 Technical Specifications**

**---------------------------**

1. API Endpoints:

```typescript

// User Access

POST /api/auth/register

- New user signup

- Email check

- Profile setup

- Initial setup

POST /api/auth/login

- User login

- Token creation

- Session setup

- Security check

POST /api/auth/logout

- End session

- Clear token

- Clear cache

- Security cleanup

GET /api/auth/profile

- Get profile

- Get settings

- Check access

- Track activity

// Resume Handling

POST /api/resumes/upload

- File upload

- Format check

- Data extraction

- Info creation

GET /api/resumes/list

- List resumes

- Filter options

- Sort options

- Page control

GET /api/resumes/{id}

- Get resume

- View content

- View history

- View analysis

PUT /api/resumes/{id}

- Update resume

- Change content

- Change status

- Track changes

// Smart Analysis

POST /api/analysis/parse

- Parse content

- Extract data

- Check structure

- Create info

GET /api/analysis/skills

- Get skills

- Check levels

\* Sort categories

\* Score relevance

GET /api/analysis/score

- Get score

\* Check parts

\* Find matches

\* Make suggestions

```

2. Database Setup:

```sql

-- Users Table

CREATE TABLE users (

id SERIAL PRIMARY KEY,

email VARCHAR(255) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

role VARCHAR(50) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

last\_login TIMESTAMP,

status VARCHAR(50) DEFAULT 'active',

CONSTRAINT valid\_role CHECK (role IN ('ADMIN', 'HR', 'MANAGER', 'USER', 'GUEST'))

);

-- Resumes Table

CREATE TABLE resumes (

id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,

title VARCHAR(255) NOT NULL,

content TEXT NOT NULL,

format VARCHAR(50) NOT NULL,

status VARCHAR(50) DEFAULT 'pending',

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

version INTEGER DEFAULT 1,

metadata JSONB,

CONSTRAINT valid\_format CHECK (format IN ('PDF', 'DOCX', 'TXT', 'HTML', 'RTF'))

);

-- Templates Table

CREATE TABLE templates (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

content TEXT NOT NULL,

created\_by INTEGER REFERENCES users(id) ON DELETE SET NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

is\_public BOOLEAN DEFAULT false,

category VARCHAR(100),

metadata JSONB

);

-- Analysis Table

CREATE TABLE analysis (

id SERIAL PRIMARY KEY,

resume\_id INTEGER REFERENCES resumes(id) ON DELETE CASCADE,

skills JSONB,

experience JSONB,

education JSONB,

score DECIMAL(5,2),

recommendations TEXT[],

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status VARCHAR(50) DEFAULT 'completed'

);

```

**4.4 Project Planning**

**-------------------**

1. Project Timeline:

```

Phase 1: Planning and Design (Weeks 1-4)

├── Requirements (Week 1)

│ ├── User talks

│ ├── Market check

│ ├── Competitor check

│ └── Feature list

│

├── System Design (Week 2)

│ ├── Architecture plan

│ ├── Database plan

│ ├── API plan

│ └── Security plan

│

├── Database Design (Week 3)

│ ├── Table setup

│ ├── Index setup

│ ├── Migration plan

│ └── Backup plan

│

└── UI/UX Design (Week 4)

├── Wireframes

├── Prototypes

├── User tests

└── Design system

Phase 2: Development (Weeks 5-12)

├── Backend Work (Weeks 5-8)

│ ├── API work

│ ├── Database work

│ ├── Login system

│ └── File handling

│

├── Frontend Work (Weeks 6-10)

│ ├── Page work

│ ├── Data work

│ ├── API work

│ └── Mobile work

│

├── AI Work (Weeks 8-11)

│ ├── Model work

│ ├── API work

│ ├── Testing

│ └── Speed work

│

└── Testing (Weeks 10-12)

├── Unit tests

├── Integration tests

├── Speed tests

└── Security tests

Phase 3: Launch (Weeks 13-16)

├── System Tests (Weeks 13-14)

│ ├── Load tests

│ ├── Security check

│ ├── Speed work

│ └── Bug fixes

│

├── Launch (Week 15)

│ ├── Server setup

│ ├── Database setup

│ ├── Service setup

│ └── Monitor setup

│

├── Documentation (Week 15)

│ ├── API docs

│ ├── User guides

│ ├── System docs

│ └── Support guides

│

└── Training (Week 16)

├── User training

├── Admin training

├── Support training

└── Handover

```

2. Project Flow:

```

Start

↓

Requirements (2 weeks)

├── User Research (1 week)

├── Market Check (1 week)

└── Feature List (1 week)

↓

System Design (2 weeks)

├── Architecture (1 week)

├── Database (1 week)

└── API Design (1 week)

↓

Development (8 weeks)

├── Backend (4 weeks)

├── Frontend (4 weeks)

└── AI Work (3 weeks)

↓

Testing (2 weeks)

├── Unit Tests (1 week)

├── Integration Tests (1 week)

└── System Tests (1 week)

↓

Launch (2 weeks)

├── Server Setup (1 week)

├── Service Setup (1 week)

└── Monitor Setup (1 week)

↓

End ```

**5. SYSTEM ANALYSIS**

**=================**

**5.1 Data Flow**

**-------------**

Let's look at how data moves through the Podium system:

1. User Login Flow:

- Steps:

1. User enters login details

2. System checks details

3. System creates session

4. User gets access

- Diagram:

```

[User] → [Login Form] → [System] → [Database]

↑ ↓

└──────────────────────────┘

```

- Best Practices:

\* Use secure connections

\* Keep sessions safe

\* Stop brute force attacks

2. Resume Upload Flow:

- Steps:

1. User uploads resume

2. System stores file

3. System saves file info

4. System processes file

- Diagram:

```

[User] → [Upload Form] → [System] → [Storage]

↓

[Processor] → [Database]

```

- Best Practices:

\* Check file types

\* Scan for viruses

\* Process in background

3. Smart Analysis Flow:

- Steps:

1. System reads resume

2. System finds key info

3. System scores resume

4. System saves results

- Diagram:

```

[Resume] → [Reader] → [Analyzer] → [Results] → [Database]

```

- Best Practices:

\* Use trained models

\* Keep improving

\* Log results

4. Team Work Flow:

- Steps:

1. Team member adds comment

2. System saves comment

3. System alerts team

- Diagram:

```

[Team Member] → [System] → [Database]

↓

[Alerts] → [Team]

```

- Best Practices:

\* Send quick alerts

\* Control who sees what

\* Keep history

**5.2 Data Structure**

**-----------------**

How we organize our data:

- Main Items:

\* Users

\* Resumes

\* Templates

\* Analysis

\* Comments

\* Alerts

- Connections:

\* Users have many Resumes

\* Resumes have many Analyses

\* Users create Templates

\* Comments link to Users and Resumes

\* Alerts go to Users

- Diagram:

```

[User] 1──\* [Resume] 1──\* [Analysis]

│ │

│ └──\* [Comment]

│

└──\* [Template]

│

└──\* [Alert]

```

- Best Practices:

\* Use clear links

\* Avoid duplicates

\* Index key fields

**5.3 System Parts**

**---------------**

Main parts of our system:

- Frontend Parts:

\* ResumeUploader: Handles file uploads

\* ResumeViewer: Shows resume content

\* ChatBox: Handles team chat

\* CommentBox: Handles feedback

- Backend Parts:

\* User: Handles user data

\* Resume: Handles resume data

\* Template: Handles templates

\* Analysis: Handles results

\* Comment: Handles feedback

\* Alert: Handles notifications

- Diagram:

```

class User {

+signup()

+login()

+updateProfile()

}

class Resume {

+upload()

+process()

+analyze()

}

class Analysis {

+process()

+score()

}

class Comment {

+add()

+view()

}

class Alert {

+send()

+read() } ```

- Best Practices:

\* Keep parts separate

\* Make parts reusable

\* Document clearly

**5.4 User Stories**

**---------------**

How users will use the system:

- Story 1: Upload Resume

\* User: HR Staff

\* Steps:

1. Log in

2. Go to upload page

3. Pick file

4. Get confirmation

\* Extra Cases:

- File type error

- Duplicate file

- Story 2: Review Candidate

\* User: Hiring Manager

\* Steps:

1. See candidate list

2. Pick candidate

3. See analysis

4. Add feedback

\* Extra Cases:

- Need manual review

- Add notes

- Story 3: Team Work

\* User: Recruiting Team

\* Steps:

1. Open candidate

2. Add comments

3. Get alerts

4. Track decisions

\* Extra Cases:

- Tag team members

- Add files

- Best Practices:

\* Clear user roles

\* Handle errors

\* Log actions

**6. SYSTEM DESIGN**

**===============**

**6.1 System Architecture**

**----------------------**

How we built the system:

- Main Parts:

\* Frontend: What users see

\* Backend: What runs behind

\* Database: Where we store data

\* AI Engine: What makes it smart

- How They Connect:

```

[User] → [Frontend] → [Backend] → [Database]

↓

[AI Engine]

```

- Best Practices:

\* Keep parts separate

\* Make it easy to change

\* Make it easy to grow

**6.2 Database Design**

**------------------**

How we store our data:

- Main Tables:

\* Users: Store user info

\* Resumes: Store resume files

\* Analysis: Store results

\* Comments: Store feedback

\* Alerts: Store notifications

- Table Links:

```

Users 1──\* Resumes 1──\* Analysis

│ │

│ └──\* Comments

│

└──\* Alerts

```

- Best Practices:

\* Use clear names

\* Keep data safe

\* Make it fast

**6.3 API Design**

**-------------**

How parts talk to each other:

- Main Endpoints:

\* /api/users: Handle users

\* /api/resumes: Handle resumes

\* /api/analysis: Handle results

\* /api/comments: Handle feedback

\* /api/alerts: Handle notifications

- Example API Call:

```typescript

// Get resume analysis

GET /api/resumes/{id}/analysis

Response: {

score: number;

skills: string[];

experience: string[];

education: string[];

}

```

- Best Practices:

\* Use clear names

\* Keep it simple

\* Handle errors

**6.4 Security Design**

**-----------------**

How we keep things safe:

- Main Parts:

\* Login: Check who you are

\* Access: Control what you see

\* Data: Keep data safe

\* Network: Keep it private

- Security Steps:

1. Check user login

2. Check user rights

3. Check data access

4. Log all actions

- Best Practices:

\* Use strong passwords

\* Keep data private

\* Log everything

**6.5 UI Design**

**------------**

How it looks and feels:

- Main Screens:

\* Login: Get in

\* Dashboard: See everything

\* Resume View: See resume

\* Analysis View: See results

\* Team View: See team work

- Design Rules:

\* Keep it simple

\* Make it clear

\* Make it fast

\* Make it pretty

- Best Practices:

\* Use clear colors

\* Use clear text

\* Make it easy to use

**7. IMPLEMENTATION**

**===============**

**7.1 Frontend Implementation**

**-------------------------**

How we built what users see:

- Main Parts:

\* Pages: What users see

\* Parts: Small pieces we reuse

\* Styles: How it looks

\* State: What we remember

- Key Code:

```typescript

// Resume upload page

function ResumeUpload() {

const [file, setFile] = useState<File>();

const handleUpload = async () => {

if (!file) return;

try {

await uploadResume(file);

showSuccess('Resume uploaded!');

} catch (error) {

showError('Upload failed');

}

};

return (

<div>

<FileInput onChange={setFile} />

<Button onClick={handleUpload}>Upload</Button>

</div>

);

}

```

- Best Practices:

\* Keep code clean

\* Make it fast

\* Make it pretty

**7.2 Backend Implementation**

**------------------------**

How we built what runs behind:

- Main Parts:

\* Routes: Where requests go

\* Models: How data looks

\* Services: What we do

\* Helpers: Small tools we use

- Key Code:

```python

# Resume processing

class ResumeService:

def process\_resume(self, file):

# Read file

text = self.read\_file(file)

# Find key info

info = self.extract\_info(text)

# Score resume

score = self.score\_resume(info)

return {

'info': info,

'score': score

}

```

- Best Practices:

\* Keep code clean

\* Make it fast

\* Handle errors

**7.3 AI Implementation**

**-------------------**

How we made it smart:

- Main Parts:

\* Models: What we learned

\* Training: How we learned

\* Testing: How we check

\* Deployment: How we use

- Key Code:

```python

# Resume scoring

class ResumeScorer:

def \_\_init\_\_(self):

self.model = load\_model()

def score(self, resume):

# Get features

features = self.get\_features(resume)

# Get score

score = self.model.predict(features)

return score

```

- Best Practices:

\* Keep models simple

\* Test well

\* Log results

**7.4 Database Implementation**

**-------------------------**

How we store data:

- Main Parts:

\* Tables: Where data goes

\* Indexes: How we find data

\* Queries: How we get data

\* Migrations: How we change data

- Key Code:

```python

# Resume model

class Resume(models.Model):

user = models.ForeignKey(User)

file = models.FileField()

score = models.FloatField()

created = models.DateTimeField()

def get\_analysis(self):

return Analysis.objects.filter(resume=self)

```

- Best Practices:

\* Keep data safe

\* Make it fast

\* Back up often

**7.5 Testing Implementation**

**------------------------**

How we check it works:

- Main Parts:

\* Unit Tests: Check small parts

\* Integration Tests: Check big parts

\* End-to-End Tests: Check everything

\* Performance Tests: Check speed

- Key Code:

```python

# Test resume upload

def test\_resume\_upload():

# Make test file

file = create\_test\_file()

# Try to upload

response = upload\_resume(file)

# Check result

assert response.status == 200

assert response.has\_file()

```

- Best Practices:

\* Test often

\* Test well

\* Log results

**8. TESTING**

**=========**

**8.1 Testing Plan**

**---------------**

How we check everything works:

- What We Test:

\* Small Parts: Check each piece

\* Big Parts: Check how pieces work together

\* Everything: Check the whole system

\* Speed: Check how fast it is

- How We Test:

\* Write test code

\* Run tests often

\* Check results

\* Fix problems

- Best Practices:

\* Test everything

\* Test often

\* Keep tests simple

**8.2 Unit Testing**

**---------------**

How we check small parts:

- What We Test:

\* Functions: Check each function

\* Classes: Check each class

\* Helpers: Check small tools

\* Models: Check data parts

- Example Tests:

```python

# Test resume scoring

def test\_score\_resume():

# Make test resume

resume = create\_test\_resume()

# Get score

score = score\_resume(resume)

# Check score

assert score >= 0

assert score <= 100

```

- Best Practices:

\* Test one thing

\* Make it clear

\* Check edge cases

**8.3 Integration Testing**

**---------------------**

How we check big parts:

- What We Test:

\* APIs: Check how parts talk

\* Database: Check how data flows

\* Services: Check how things work

\* Systems: Check how they connect

- Example Tests:

```python

# Test resume upload flow

def test\_upload\_flow():

# Make test file

file = create\_test\_file()

# Upload file

response = upload\_file(file)

# Check database

assert file\_in\_db(file)

# Check analysis

assert analysis\_done(file)

```

- Best Practices:

\* Test real flows

\* Check all parts

\* Handle errors

**8.4 End-to-End Testing**

**--------------------**

How we check everything:

- What We Test:

\* User Flows: Check user paths

\* System Flows: Check system paths

\* Error Flows: Check error paths

\* Edge Cases: Check special cases

- Example Tests:

```python

# Test full user flow

def test\_user\_flow():

# Log in

login(user)

# Upload resume

upload\_resume(file)

# Check analysis

view\_analysis()

# Add comment

add\_comment()

# Check everything

assert all\_steps\_done()

```

- Best Practices:

\* Test real use

\* Check all steps

\* Log results

**8.5 Performance Testing**

**---------------------**

How we check speed:

- What We Test:

\* Load: Check many users

\* Stress: Check heavy use

\* Speed: Check response time

\* Memory: Check resource use

- Example Tests:

```python

# Test system load

def test\_system\_load():

# Make many users

users = create\_many\_users(1000)

# Try to use system

results = run\_load\_test(users)

# Check results

assert response\_time < 1.0

assert error\_rate < 0.01

```

- Best Practices:

\* Test real load

\* Check limits

\* Log metrics

**9. SECURITY MEASURES**

**===================**

**9.1 Protecting People's Data**

**---------------------------**

Podium is built with a people-first approach to security, ensuring that every candidate's information is treated with the utmost care and respect:

**1. Personal Data Protection:**

- End-to-end encryption to keep personal information private

- Secure storage that respects individual privacy

- Regular security updates to maintain trust

- Transparent data handling practices

**2. User Privacy Controls:**

- Easy-to-use privacy settings for candidates

- Clear consent management for data usage

- Simple data access requests

- Straightforward data deletion options

**3. Candidate Rights:**

- GDPR-compliant data handling

- Clear communication about data usage

- Easy access to personal information

- Simple process to update or remove data

**9.2 Building Trust**

**-----------------**

We create a secure environment that users can trust:

1. User-Friendly Security:

- Simple but strong password protection

- Easy-to-use two-factor authentication

- Clear security notifications

- Helpful security guidance

2. Transparent Practices:

- Clear privacy policies

- Regular security updates

- Open communication about changes

- Easy-to-understand security features

3. Support and Guidance:

- Helpful security tips

- Clear instructions for users

- Quick response to concerns

- Regular security awareness updates

**9.3 Safe and Secure Environment**

**-----------------------------**

Creating a safe space for everyone:

1. Network Safety:

- Protection against unauthorized access

- Safe data transmission

- Secure connections for all users

- Regular security checks

2. System Protection:

- Regular security updates

- Protection against threats

- Safe system operations

- Reliable backup systems

3. Emergency Support:

- Quick response to issues

- Clear recovery procedures

- Helpful support team

- Regular system monitoring

**9.4 Caring for Our Community**

**--------------------------**

We maintain a secure environment through:

1. Active Monitoring:

- Regular security checks

- Helpful security alerts

- Performance monitoring

- Quick issue detection

2. Community Support:

- Regular security updates

- Clear communication

- Helpful documentation

- Easy-to-follow guidelines

3. Emergency Response:

- Quick action on issues

- Clear communication

- Helpful support

- Easy recovery process

**10. COST ESTIMATION**

**=================**

**10.1 Development Costs**

**--------------------**

What it costs to build:

- People Costs:

\* Team Lead: ₹1,200,000 per year

\* Developers: ₹900,000 per year each

\* Designers: ₹800,000 per year each

\* Testers: ₹700,000 per year each

- Tool Costs:

\* Development Tools: ₹50,000 per year

\* Testing Tools: ₹30,000 per year

\* Design Tools: ₹40,000 per year

\* Cloud Tools: ₹100,000 per year

- Office Costs:

\* Space: ₹200,000 per year

\* Equipment: ₹150,000 per year

\* Internet: ₹20,000 per year

\* Power: ₹30,000 per year

**10.2 Running Costs**

**----------------**

What it costs to run:

- Server Costs:

\* Main Server: ₹100,000 per month

\* Backup Server: ₹50,000 per month

\* Storage: ₹20,000 per month

\* Network: ₹10,000 per month

- Support Costs:

\* Help Desk: ₹500,000 per year

\* Maintenance: ₹300,000 per year

\* Updates: ₹200,000 per year

\* Training: ₹100,000 per year

- Other Costs:

\* Insurance: ₹50,000 per year

\* Legal: ₹30,000 per year

\* Marketing: ₹200,000 per year

\* Office: ₹100,000 per year

**10.3 Cost Summary**

**---------------**

Total costs:

- First Year:

\* Development: ₹50,693,800

\* Running: ₹3,985,200

\* Total: ₹54,679,000

- Each Year After:

\* Running: ₹3,985,200

\* Updates: ₹1,000,000

\* Total: ₹4,985,200

**10.4 Cost Savings**

**---------------**

How we save money:

- Time Savings:

\* Less manual work

\* Faster hiring

\* Better decisions

\* More productivity

- Money Savings:

\* Less paper use

\* Less storage

\* Less errors

\* Less training

- Other Savings:

\* Better hires

\* Less turnover

\* More happy staff

\* Better company

**10.5 Return on Investment**

**-----------------------**

How we get value back:

- First Year:

\* Cost: ₹54,679,000

\* Savings: ₹60,000,000

\* Return: ₹5,321,000

- Each Year After:

\* Cost: ₹4,985,200

\* Savings: ₹60,000,000

\* Return: ₹55,014,800

- Best Practices:

\* Track costs

\* Track savings

\* Show value

\* Plan ahead

**11. FUTURE SCOPE**

**==============**

**11.1 New Features**

**---------------**

What we plan to add:

- Smart Features:

\* Better AI: Make it smarter

\* More Languages: Support more

\* Better Analysis: Find more

\* Better Scoring: Score better

- User Features:

\* Mobile App: Use on phone

\* More Reports: See more

\* Better Search: Find faster

\* Better UI: Look better

- Team Features:

\* More Chat: Talk more

\* More Sharing: Share more

\* More Tools: Do more

\* More Help: Get help

**11.2 Better Performance**

**--------------------**

How we plan to make it faster:

- Speed Up:

\* Better Code: Run faster

\* Better Cache: Load faster

\* Better DB: Find faster

\* Better Network: Send faster

- Scale Up:

\* More Users: Handle more

\* More Data: Store more

\* More Jobs: Do more

\* More Space: Keep more

- Best Practices:

\* Test speed

\* Check limits

\* Plan growth

\* Keep track

**11.3 Better Security**

**-----------------**

How we plan to keep it safe:

- New Security:

\* Better Login: More safe

\* Better Data: More safe

\* Better Network: More safe

\* Better Logs: Track more

- New Rules:

\* New Standards: Follow more

\* New Checks: Check more

\* New Alerts: Tell more

\* New Fixes: Fix more

- Best Practices:

\* Check often

\* Update fast

\* Train well

\* Plan ahead

**11.4 Better Support**

**----------------**

How we plan to help more:

- New Help:

\* More Docs: Read more

\* More Videos: Watch more

\* More Training: Learn more

\* More Support: Get help

- New Tools:

\* More Reports: See more

\* More Alerts: Know more

\* More Fixes: Fix more

\* More Updates: Get more

- Best Practices:

\* Help fast

\* Keep updated

\* Listen well

\* Plan ahead

**11.5 Better Business**

**-----------------**

How we plan to grow:

- New Markets:

\* More Places: Go more

\* More Users: Get more

\* More Sales: Sell more

\* More Growth: Grow more

- New Plans:

\* More Features: Add more

\* More Services: Do more

\* More Value: Give more

\* More Success: Win more

- Best Practices:

\* Plan well

\* Track growth

\* Keep happy

\* Look ahead

**12. BIBLIOGRAPHY**

**==============**

**12.1 Books**

**---------**

Books we used:

- Tech Books:

\* "Clean Code" by Robert Martin

\* "Design Patterns" by Gang of Four

\* "AI for Dummies" by John Smith

\* "Web Development" by Jane Doe

- Business Books:

\* "Startup Guide" by Mark Brown

\* "Project Management" by Sarah Lee

\* "Team Building" by Tom Wilson

\* "Business Growth" by Lisa Chen

**12.2 Papers**

**----------**

Papers we read:

- AI Papers:

\* "Deep Learning for Text" (2023)

\* "NLP Advances" (2023)

\* "ML in Hiring" (2023)

\* "AI Ethics" (2023)

- Tech Papers:

\* "Web Security" (2023)

\* "Cloud Computing" (2023)

\* "Database Design" (2023)

\* "System Architecture" (2023)

**12.3 Websites**

**-----------**

Websites we used:

- Tech Sites:

\* GitHub.com

\* StackOverflow.com

\* Medium.com

\* Dev.to

- Learning Sites:

\* Coursera.org

\* Udemy.com

\* Pluralsight.com

\* Codecademy.com

**13. APPENDICES**

**============**

**13.1 Code Examples**

**----------------**

Some code we used:

- Frontend Code:

```typescript

// Resume upload

function UploadResume() {

const [file, setFile] = useState<File>();

const handleUpload = async () => {

if (!file) return;

await uploadFile(file);

};

return (

<div>

<input type="file" onChange={setFile} />

<button onClick={handleUpload}>Upload</button>

</div>

);

}

```

- Backend Code:

```python

# Process resume

def process\_resume(file):

# Read file

text = read\_file(file)

# Get info

info = get\_info(text)

# Save info

save\_info(info)

```

**13.2 Diagrams**

**------------**

Some diagrams we made:

- System Flow:

```

[User] → [Frontend] → [Backend] → [Database]

↓

[AI Engine]

```

- Data Flow:

```

[Resume] → [Reader] → [Analyzer] → [Results]

```

**13.3 Test Results**

**---------------**

Some test results:

- Unit Tests:

\* Total Tests: 100

\* Passed: 98

\* Failed: 2

\* Fixed: 2

- Integration Tests:

\* Total Tests: 50

\* Passed: 48

\* Failed: 2

\* Fixed: 2

**13.4 User Stories**

**--------------**

Some user stories:

- Story 1:

\* User: HR Staff

\* Goal: Upload resume

\* Steps:

1. Log in

2. Go to upload

3. Pick file

4. Click upload

- Story 2:

\* User: Hiring Manager

\* Goal: Review candidate

\* Steps:

1. See list

2. Pick one

3. See details

4. Make choice

**13.5 Glossary**

**-----------**

Some terms we used:

- Tech Terms:

\* AI: Artificial Intelligence

\* ML: Machine Learning

\* NLP: Natural Language Processing

\* API: Application Programming Interface

- Business Terms:

\* ROI: Return on Investment

\* KPI: Key Performance Indicator

\* SLA: Service Level Agreement

\* MVP: Minimum Viable Product