CS261: Software Engineering Project Planning and Design Report - Group 23

Harry Fallows, Simon Heap, Bhavik Makwana, Pavnish Mohan, Antonia Newey

Version: 1.4.r
Date created: 14-01-19
Last modified: 12–03–2019

Contents

1 Introduction

The purpose of this document is to provide a design for the deliverable, an automatic applicant evaluation system. This document is intended to be read by the client, project manager, initial developers, testers and developers maintaining the system. It will clarify the structure of the software, resources/technologies used, testing against the requirements, and the project plan.

2 Design Considerations

2.1 Extensibility

The solution is required to be extensible to deal with the fluctuation in the volume of candidates throughout an application window. Building the system on a cloud based infrastructure will allow it to remain responsive as it can easily scale to the client's demands.

2.2 Robustness

Robustness primarily refers to validating user input throughout the system. The sample dataset provided contains only valid data, however data input in production may not.

To filter user input for the machine learning (ML) model, everything will be converted to lowercase in order to prevent the model evaluating the same thing as more than one entity. For example, 'university of warwick' and 'University of Warwick' should be evaluated as the same entity.

The previous employment, hobbies, languages, and skills fields are all able to be left blank. However, certain fields such as name will be mandatory to fill in during the application and must be validated at the input stage by both the client- and server-side systems.

University attended, degree level, and degree type will also be validated through the use of a drop-down list client-side and a whitelist server-side.

2.3 Reliability

To ensure the system will perform under everyday conditions, it will be developed using a test-driven ideology. This ensures all components are working correctly through extensive unit and integration tests; this will be done using pytest.

Using a cloud based infrastructure will increase the system's reliability as the server's capacity can be easily expanded when the system is under heavy load. This reduces the finite low-level control the client has over the system's architecture but helps to abstract the issues associated with setting up and expanding a server. Modern cloud service providers are also extremely robust and reliable, meaning it is unlikely the service would fail.

2.4 Correctness

Correctness refers to how accurately the solution meets the customer's requirements. Creating an exhaustive set of client requirements and converting them into clear and succinct developer requirements ensures the final solution will closely meet the deliverable requested by the customer.

2.5 Compatibility and portability

The deliverable will not require installation by candidates as it will be accessible from a web browser.

The web app will have a separate login portal for the client. In addition to this, the client will have the option to download a mobile app to view the information on a portable system.

The backend of the system will require an initial setup for the live application which will need to be maintained. Installation of the system could be made easier by putting it into a container using Docker [1].

2.6 Modularity and reuse

The system will be modular. The backend will be a RESTful API which will be accessed by the frontend from both the web app and the mobile app to ensure consistency across all platforms. The API itself will be modular, splitting each feature into a function so it can be easily reused across the system.

Various libraries and frameworks will be used to speed up development such as Flask, Bulma, jQuery, and existing code libraries.

2.7 Security

Minimal security measures will be implemented in the final deliverable because the project specification indicated it is out of scope. Security still must be considered to allow future iterations of the system to comply with GDPR[2].

2.8 Fault-tolerance

A cloud-based infrastructure allows for system fault-tolerance. If one of the servers fails, another can take its place. Data redundancy will be introduced to the system to decrease the risk of data loss and to minimise system downtime.

3 Technical Description

3.1 Relationships between systems

The deliverable will consist of a backend system built using a MongoDB database and a RESTful API built in Flask; this will provide support for both a web and mobile app. The frontend components will interact with the API to send information and display received data. This modular structure will reduce code redundancy and ensure that it is easily extendable. A simple diagram outlining the relationship between these entities is shown in Figure 1.

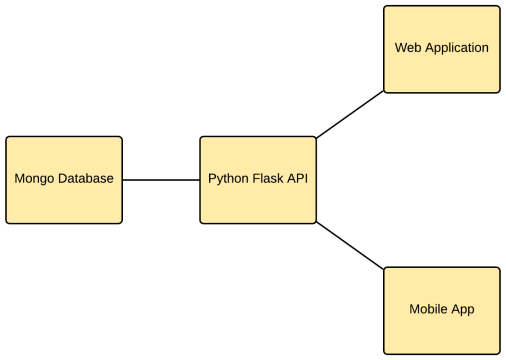


Figure 1: System Context Model

3.2 Frontend

3.2.1 Web App

The web app will use HTML, CSS, and JavaScript to build an interface that all users will see. This interface will be simple and will do as little data processing as possible; the API will handle all data processing. A web application with local processing was considered, however backend processing would help increase the modularity of the system, allowing for easy reuse. A simple frontend application would also aid in the security of the system, since it would help confine most security responsibilities to the backend API.

Bulma will be used for its clear designs and lightweight framework, with numerous optional extensions; this will make the system easier to design and build, and increase its reliability.

3.2.2 Mobile App

The mobile app will also make use of the API to handle most of its data processing, but with a reduced feature set. Both Android and iOS were considered as platforms for the mobile app. However, Android will be the priority platform due to its flexibility and wide use.

3.3 Backend

3.3.1 API

The backend API will be built using Flask, a Python microframework, for ease of reading and maintenance. Additionally, the Flask framework allows for the use of up-to-date security extensions. The PyMongo library will allow the MongoDB database to interact with the API.

When processing the dataset to determine the best candidate for the job, the system will use a number of classes to store, load, and filter through the data appropriately. To express this, a simple class diagram can be seen in Figure 2.

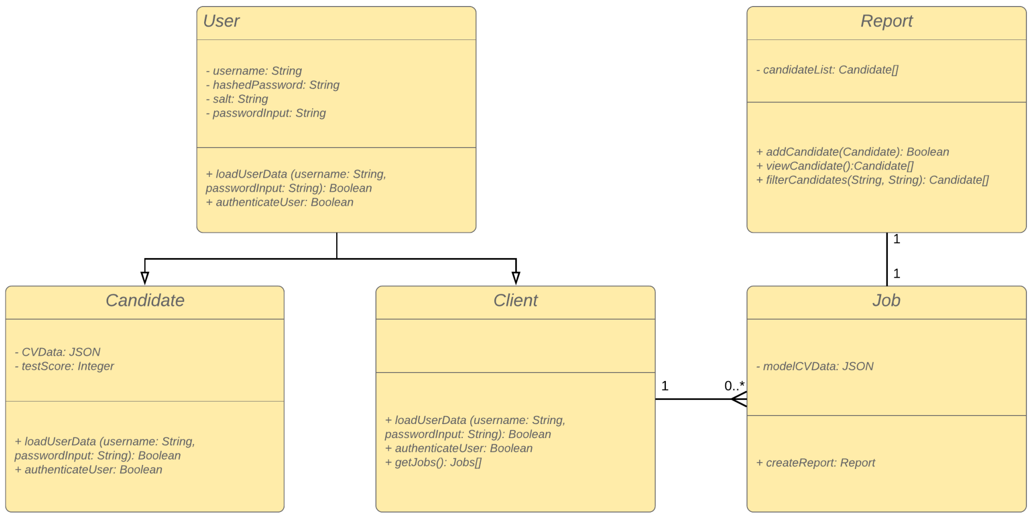

User
- username: String
- hashedPassword: String
- salt: String
- passwordInput: String
+ loadUserData (username: String, passwordInput: String): Boolean
+ authenticateUser: Boolean
Report
- candidateList: Candidate[]
+ addCandidate(Candidate): Boolean
+ viewCandidate():Candidate[]
+ filterCandidates(String, String): Candidate[]
1
1
Candidate
- CVData: JSON
- testScore: Integer
+ loadUserData (username: String, passwordInput: String): Boolean
+ authenticateUser: Boolean
Client
+ loadUserData (username: String, passwordInput: String): Boolean
+ authenticateUser: Boolean
+ getJobs(): Jobs[]
1
0..*
Job
- modelCVData: JSON
+ createReport: Report

Figure 2: System Class Diagram

3.3.2 Database

MongoDB databases store their data in JSON files, so is perfect for handling the CV data provided by the client. An entity relationship diagram showing the collections used can be seen in Figure 3.

Because the database will be storing personal data, the system will need to abide by the 6 GDPR privacy principles [2]. The system will store data lawfully, accurately, for a finite time, and purely for the purpose of hiring. The data's integrity and confidentiality will be ensured when storing it, and users will be allowed to request the removal of their data from the system.

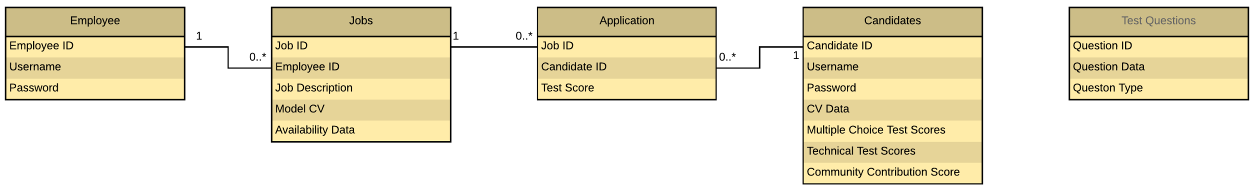


Figure 3: System Entity Relationship Diagram

3.3.3 Machine Learning

The problem statement is best framed as an 8-dimensional regression [3], which suggests the best candidates based on contents from their CV (A-levels, Degree Level, Degree Qualification, Languages Known, Previous Employment, and Skills), their test scores, and their contributions to open-source projects. Initially, it will be trained using Degree Qualification, Previous Employment, and Languages Known for simplicity.

The model will choose those candidates whose CV data most closely matches an exemplar CV derived from the Job Description by using cosine similarity[4]. The dataset provided consists of 100,000 unlabelled example CVs in a JSON format. For test scores and community collaboration random scores will be appended to each CV to provide the complete dataset for each candidate.

As the dataset is unlabelled online[5], reinforcement learning[6] will assist the model in understanding whether the candidates it selects meet the clients requirements for the position or not.

The data will need to be prepared as ML models cannot accept text as input, so the CV data will need to be converted into a vector using a method such as one-hot encoding.

During development common python libraries for machine learning such as scypy, pandas, matplotlib, etc. will be used to aid in preparing and visualising the data, alongside the tensorflow and keras frameworks to develop the neural net.

3.4 System Evolution

3.4.1 Assumptions

The client has requested for a self-learning system to be created to help assess candidates. Because of this, the system should be able to evolve on its own, to a certain extent, to select the most suitable candidates for each available position.

To accommodate for the changes in the client's needs, they must give feedback to assist the model's learning by determining if the candidates selected meet their requirements.

The client has stated teamwork and collaboration as a key area of interest when assessing candidates. As well as this, they would like candidates with good experience, programming skills, and employability skills. For each role they will be able to specify the minimum and preferred competency in each skill, and the ML model will learn over time to select candidates based on this.

It is also assumed the client has a way to provide feedback for reinforced learning and a sufficient rate, i.e. a bot or human employees.

3.4.2 System architecture evolution

As the volume of candidates is unknown, the system's architecture must be able to accommodate for fluctuations in application numbers. Also, as the deliverable is being built under the assumption that it is a stand-alone service from the client's internal systems, it would be best hosted on a cloud platform. These are ideal because they are easily scalable and are a cost-effective solution with no up-front cost.

The client has stated storage and server capacity are not an issue, and a cloud based system does not limit the amount of training data that can be stored. The physical security of these data centres has increased drastically over the years, making it safer to store sensitive data on the cloud. Data could be encrypted if necessary, but this would have an impact on performance. As security is not an explicit concern, this will not be implemented.

The client has primarily requested for a system to handle applicants for technology roles within the company. However, if the deliverable is successful then they would like the system to be easily adapted to deal with non-technical candidates applying for non-technical roles.

The mobile app will require patches to keep up to date with the latest versions of android and iOS released to ensure it remains compatible with modern devices and to fix any potential security issues.

4 Description of processes and activities

To outline the processes and plan the interactions between the user and the interface, multiple charts and diagrams were created which can be seen in Figures 4, 5, 6 and 7.

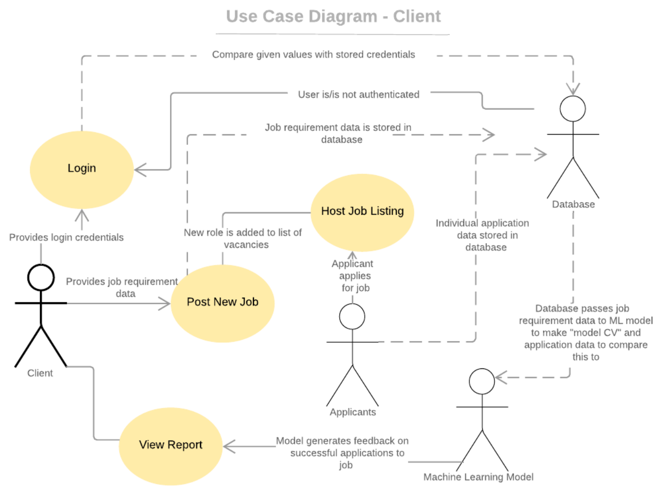

Use Case Diagram - Client
Compare given values with stored credentials
User is/is not authenticated
Job requirement data is stored in database
Login
Database
Host Job Listing
Provides login credentials
New role is added to list of vacancies
Individual application data stored in database
Applicant applies for job
Provides job requirement data
Post New Job
Client
Database passes job requirement data to ML model to make "model CV" and application data to compare this to
Applicants
View Report
Model generates feedback on successful applications to job
Machine Learning Model

Figure 4: Client Use Case Diagram

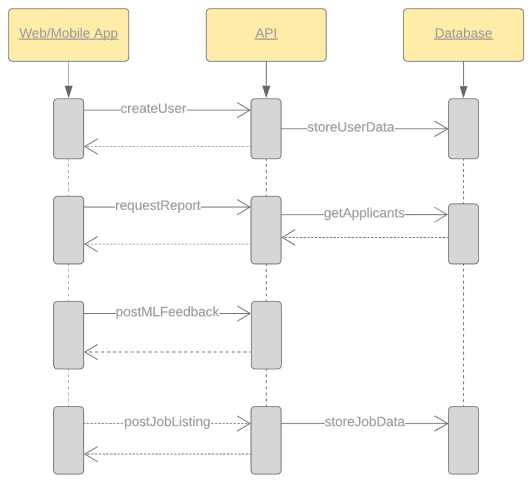


Figure 5: Client Sequence Diagram

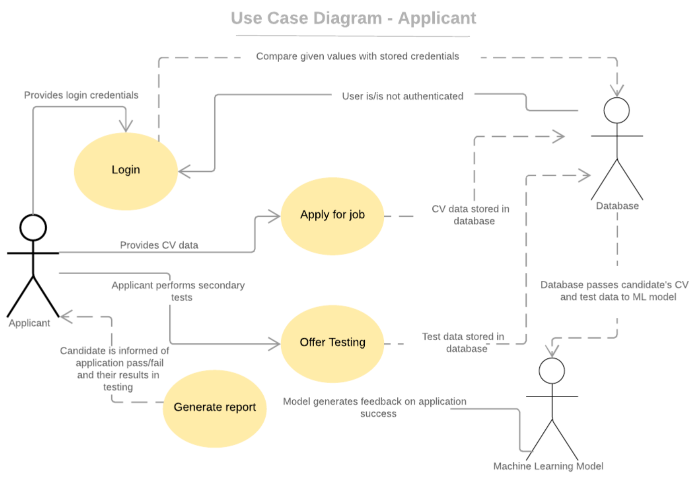

Use Case Diagram - Applicant
Compare given values with stored credentials
Provides login credentials
User is/is not authenticated
Login
Database
Apply for job
CV data stored in database
Applicant
Provides CV data
Applicant performs secondary tests
Database passes candidate's CV and test data to ML model
Offer Testing
Test data stored in database
Candidate is informed of application pass/fail and their results in testing
Generate report
Model generates feedback on application success
Machine Learning Model

Figure 6: Applicant Use Case Diagram

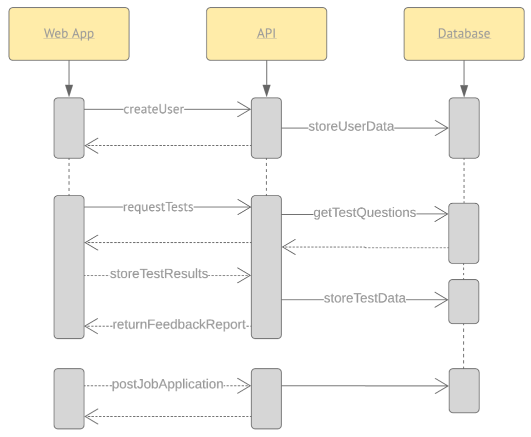


Figure 7: Applicant Sequence Diagram

5 User Requirements Design

This section details how the user requirements previously outlined for this project [7] will be met.

5.1 Functional

- **R.1** Handling application details: Candidates will submit their information using a website with a HTML form. The details required will be, at a minimum, the same as those specified in the readme.txt doc within 'CVDataset'. This information will be validated and inserted into the MongoDB database using an API.

- **R.2** Online candidate non-technical testing: Non-technical testing will also be implemented using a HTML form, with predefined questions and multiple choice answers; the user inputs will be recorded using HTML radio buttons. The answers will be checked against the model answers using a python API; scores will then be fed to the ML model.

- **R.3** Online candidate technical testing: This will be implemented using a HTML text/code field, the user's input will be sent to an API for testing. The API will run the code and check if the output(s) are valid. This information will be fed back to the candidate before they decide to submit their code. The final pass/fail status of the test will be sent to the database and used for candidate scoring in the ML model.

  - **R.3.1** The API will support multiple languages for the technical tests using the open source software OpenRank [8].

- **R.4** Community contributions: Candidates' repositories and forums will be checked and scored using open source APIs offered by the relevant services [9].

- **R.5** Candidate feedback:

  - **R.5.1, R.5.2, R.5.3** Candidates will receive quantitative feedback on their non-technical tests in the form of scores (/4) and a radar chart. This will be displayed in the report; charts will be displayed using Chart.js. For R.5.3 the minimum and preferred requirements of the job as part of the radar chart will be displayed.
  - **R.5.4** Candidates will be sent a copy the feedback through an email sent by a python API.

- **R.6** Candidate login

  - **R.6.1** Candidates will be able to log in. Candidate credentials will be stored in the database; passwords will be salted and hashed.
  - **R.6.2** Candidates' application history will be displayed on their account in the form of a web-page.

- **R.7** Application selection

  - **R.7.1** Elements of the candidates' CVs will be compared with elements of the model CV (based off the job description) using cosine similarity, these values will then be fed into a neural network.
  - **R.7.2** The client will be able to select a cut-off date using a HTML date-picker, this will be validated and saved in the database. The ML model will return its candidate list in the form of a web-page report, on this date. Alternatively a rolling report can be selected which will be an updating web-page only visible to the client.
  - **R.7.3** The client will be able to specify how many candidates should be selected by the system per position, which will be implemented as a slider/text entry field.
    - **R.7.3.1** A 're-open applications' button will be available on each report, it will create an identical job with a new date selected by the client. The new job will be inserted into the 'Jobs' collection in the database and the process will start again.
  - **R.7.4** A 'Stop' button will change the application deadline to the current date/time, returning the current selection of candidates.
  - **R.7.5** The CVs will be given scores, using cosine similarity ML model.
  - **R.7.6** Calculate the online test scores for each candidate: This will be done immediately after the tests and stored in the 'Application' table.
  - **R.7.7** Give candidates a score (/10) for their community contributions: This will be done by using the relevant open source APIs (depending on the platform). This will be stored within the 'Candidate' table.
  - **R.7.8** Training of the ML model will decide the weightings of R.7.5, R.7.6, R.7.7.

- **R.8** Client interface:

  - **R.8.1** A report will be generated for the client on a HTML web page listing recommended candidates with the ability to load and view a candidates CV and test scores at the press of a button.
  - **R.8.2** Clients will be able to log in. Client credentials will be stored in the database, passwords will be salted and hashed.

  - **R.8.3, R.8.4** The client will be able to create a job by entering all the traits into the system which will be stored as a model CV JSON object. A button will be created with a drop down allowing a previous job's required traits to be loaded in instantly. A Date input will allow an end date to be selected or a rolling deadline can be created.
  - **R.8.5** Upon selecting a candidate in the report, a client can choose to rate them on a scale of 1/10 using a simple slider. Additionally a button will allow a client to reject all candidates in the report and the system will provide a new selection. This will provide the feedback for reinforced learning.
  - **R.8.6** For candidates with their email stored in the system an email button will be created to allow the client to immediately contact a candidate.
  - **R.8.7** A mobile app will be created in android studio allowing a client to login to view job reports and create jobs using a simple input interface.

5.2 Non-Functional

- **R.9** Ease of Use:
  - **R.9.1** A simple website UI will be created using HTML, with a CSS/Sass framework called Bulma.
    - **R.9.1.1** Language used in the system will be non-technical, unless in reference to a job.
  - **R.9.2** The layout of the website will be familiar.
    - **R.9.2.1** The buttons and menus will be implemented in HTML, clearly labelled.
  - **R.9.3** The system will be implemented with web accessibility guidelines - Level A[10] in mind.
    - **R.9.3.1** Opposing colours will be used for text and background.
    - **R.9.3.2** Items will be coloured and labelled.
    - **R.9.3.3** Videos, if present, will be captioned and images will be briefly described.
- **R.10** The system is responsive:
  - **R.10.1** The website will be utilising the Bulma flex grid system, so that objects on a page will be resized to fit different screens.
    - **R.10.1.1** Visuals and interactive objects will remain accessible on smaller screens.
    - **R.10.1.2** Bulma automatically resizes objects in a visually appealing way.
  - **R.10.2** System performance is validated:
    - **R.10.2.1** System processing will be implemented as efficiently as possible.
  - **R.10.3** Regular updates will be performed.
    - **R.10.3.1** The development methodology is built to remain adaptive.
    - **R.10.3.2** The ML model will be implemented adaptively.

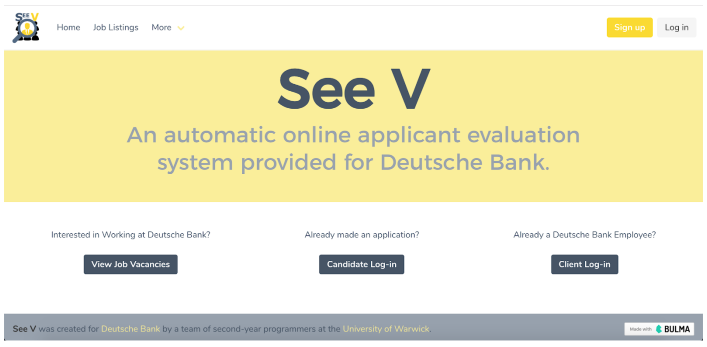
6 User Interface Design

Five main principles have been considered in designing the user interface: simplicity, visual hierarchy, comprehensive navigation, consistency, and accessibility. There are many elements of the design that overlap multiple principles.

Simplicity prevents a page from being visually overwhelming, and can in turn help meet requirements R.9.1 and R.9.2 [7] by making the interface more intuitive due to a lack of distractions. In this case, to implement a simple interface the following restrictions were made:

- A maximum of 5 colours may appear on any page
- A maximum of 3 typefaces, in up to 3 different sizes, may appear on any page
- Graphics used must be minimal, and only used when they help a user complete a task or perform a function

Creating a visual hierarchy involves organising page content so that it conveys relationships between items on a page and follows patterns that humans move their eyes in. It has been found that people from cultures that read left-to-right will most likely read in one of two patterns: F or Z[11]. The F-pattern is followed on text-heavy content, but the deliverable is unlikely to require any large blocks of text. The Z-pattern, however, can be utilised more often. For example, looking at the draft homepage shown in Figure 8:



See V

An automatic online applicant evaluation system provided for Deutsche Bank.

Interested in Working at Deutsche Bank?

Already made an application?

Already a Deutsche Bank Employee?

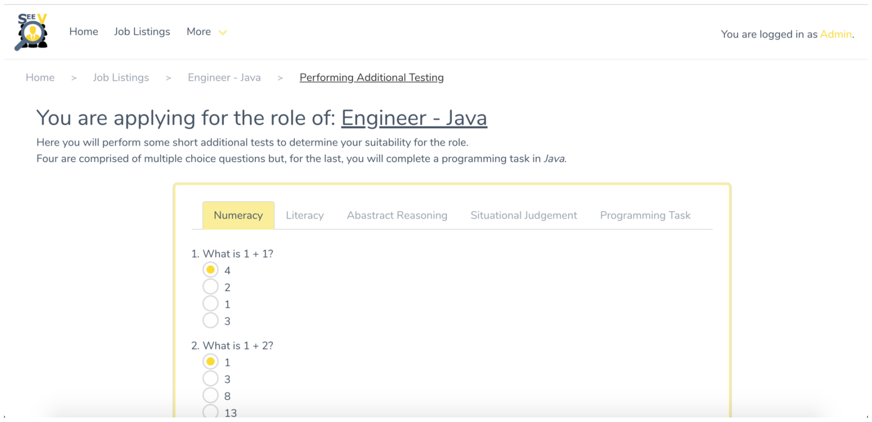
See V was created for Deutsche Bank by a team of second-year programmers at the University of Warwick.

Figure 4: Web App Homepage Design

The logo will be placed in the top left corner. Sight is then drawn to the right by a colourful button. The large 'hero' banner separates the top and bottom of the page, but is eye-catching in its own right and so the system's name will be in bold to encourage brand recognition. Along the bottom row there will be a series of buttons for the final pass over of the eye.

A comprehensive navigation system improves ease of use by limiting ways users can get "lost" on the site. The site navigation practices followed, as seen in either Figure 8 or Figure 9, include:

- Primary navigation (the nav-bar) is simple and kept at the top of the page
- Navigation is also available in the footer of the site
- Navigation is never more than four levels deep
- Breadcrumbs are provided on every page except the homepage to make people aware of their navigation trail



You are logged in as Admin.

Home > Job Listings > Engineer - Java > Performing Additional Testing

You are applying for the role of: Engineer - Java

Here you will perform some short additional tests to determine your suitability for the role.
Four are comprised of multiple choice questions but, for the last, you will complete a programming task in *Java*.

1. What is 1 + 1?
[✓ 4]
2
1
3

2. What is 1 + 2?
[✓ 1]
3
8
13

Figure 5: Web App Candidate Testing Page Design

Consistency over pages means that all pages of a site follow the same or similar layout. Consistency is an important factor of UI design as a site that changes too often can be very confusing and hard to use. A universal layout saves a user time by preventing them from having to work out where similar things are on every page.

There is more to consider when trying to making a website accessible. In one case, this might mean ensuring that the website is responsive and works on many different screen sizes. This is achieved with the help of the

CSS framework Bulma [12]. Bulma is based around the flexbox module, which redistributes and aligns page items to help prevent the loss of any content on resizing. However, accessibility may also refer to conforming to the WCAG [10]. The web app should conform to the Level A guidelines, to ensure it can be used by people who need assistive technology or those who might have trouble absorbing content in certain ways.

There are also a number of generic website design conventions that will be followed to make the deliverable more predictable for first-time users:

- The main navigation is provided at the top of a page
- The system logo is visible at the top left of a page
- The logo, when clicked, brings the user back to the homepage
- Any links on the page change colour/appearance when the user hovers over them

7 Testing

The software will be tested dynamically and statically to identify defects before deployment.

7.1 Static Testing

Static testing occurs without executing the code and is essentially verification. The use of pair programming allows one team member to be constantly evaluating the other's code. Static testing also considers attributes other than correctness, such as quality, compliance, and maintainability.

7.2 Dynamic Testing

Dynamic testing involves executing the code within small tests that will be written alongside system code as development continues. Each test is designed to check a limited amount of functionality or a unique state case. For example, a test to check whether a CV can be uploaded, or whether an application can be submitted, or a test to see if the system can handle invalid input in a form. A more complex behaviour, like filtering out candidates, may involve many unit tests for the different parts of the system it requires.

7.3 User Acceptance Testing

The final type of testing that will be carried out is user acceptance testing. Test scenarios will be decided in collaboration with the client. A small group of employees will be asked to use the new system and polled on the specification aspects outlined earlier in this document.

8 Process Documentation and Measuring Success

8.1 Plan, estimates and schedules

The development will be broken up into modules, for ease of delegation and to reduce code redundancy. This allowed for a time frame to be assigned the implementation of each section, ensuring that the sections needed for its creation were already made.

The testing of these implementations have also been taken into account so an extra two days will be set aside for the unit testing of each component before the integration tests could be run after all components had been created. This plan can be seen in the Gantt chart shown in Figure 10.

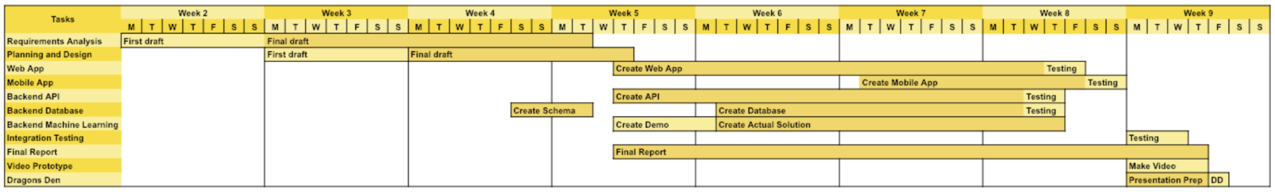


Figure 6: System Design Gantt Chart

8.2 Risk Register, monitoring and reporting

Risk	Category - Potential Damage	Solution/Mitigation
Team member falls ill or becomes unavailable	Human - Medium	Using a pair programming system, any at risk section of work can be completed by the other team member. Ensuring all team members have good knowledge of the system allows any member can pick up another's work.
Development system fails, project code progress is lost	Hardware - Large	Making regular backups to private repositories on GitHub minimises the amount of code that could be lost at a time.
Team member's personal machines fail, limiting means to write code	Hardware - Large	Ensuring the system is able to run on any machine, for example those in the Warwick Computer Science department, will mean the solution can be worked on anywhere.

Table 1: Risk Analysis Table

References

[1] Docker Documentation (version 18.09). Docker Inc.; 2019. https://docs.docker.com/.

[2] GDPR Data Protection Principles. Consultancy.uk;. https://www.consultancy.uk/news/13487/six-privacy-principles-for-general-data-protection-regulation-compliance.

[3] Problem Framing. Google; 2019. https://developers.google.com/machine-learning/problem-framing/formulate.

[4] Cosine Similarity for Vector Models;. http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/.

[5] Online Machine Learning. Wikipedia;. https://en.wikipedia.org/wiki/Online_machine_learning.

[6] Deep Reinforcement Learning. Skymind;. https://skymind.ai/wiki/deep-reinforcement-learning.

[7] Harry Fallows, Simon Heap, Bhavik Makwana, Pavnish Mohan, Antonia Newey. CS261: Software Engineering Project Requirements Analysis Report - Group 23; 2019.

[8] OpenRank Technical Test Software. Kapil Dutta;. https://github.com/coderplex/OpenRank.

[9] User Community Contribution API. GitHub;. https://developer.github.com/v3/repos/list-all-public-repositories.

[10] Eggert E AZS. How to Meet WCAG 2 (Quickref Reference). W3C;. https://www.w3.org/WAI/WCAG21/quickref/.

[11] Bigman A. 6 principles of visual hierarchy for designers. 99 Designs; 2014. https://99designs.co.uk/blog/tips/6-principles-of-visual-hierarchy/.

[12] Thomas J. Bulma Documentation (version 0.7.2). Bulma.io;. https://bulma.io/documentation/.