33rd Annual Conference of the Australasian Association for Engineering Education (AAEE 2022)
4 - 7 December 2022 – Sydney, NSW

Facilitating Scalable Vivas through Purpose-Built Software

Cornelius Paardekooper[a]; Joshua Flynn[a], Ashleigh Kirkland[a], Dylan Cuskelly[a], Alexander Gregg[a].
The University of Newcastle, Australia[a]
Corresponding Author Email: Cornelius.Paardekooper@uon.edu.au

ABSTRACT

CONTEXT

Viva Voce assessment – where students demonstrate understanding through a presentation with an assessor – is favourable for number of reasons, including inherent academic integrity, student engagement, and industry authenticity. Challengingly, these assessments scale poorly and are not efficient/sustainable for large cohorts or micro assessment. In MECH1750, an undergraduate materials design course at The University of Newcastle, scalable Viva Voce micro assessment was implemented through weekly debate-club style tutorials. These tutorials utilised moderated peer marking to aid in scalability and provide continual feedback. A cohort of 300+ students necessitated the design and deployment of a purpose-built software system for the collection, moderation, and distribution of this feedback.

PURPOSE OR GOAL

In this paper we discuss the design considerations for, implementation of and outcomes delivered by our purpose-built software solution to this problem. We outline how this solution facilitates viva-based micro assessment at scale and reduces administrative workload. We also discuss and evaluate the moderated peer marking functionality of this system. Finally, we contribute an open-source, anonymised minimal working example of the system for broader use.

APPROACH OR METHODOLOGY/METHODS

We demonstrate viability of the system and evaluate performance through usage analytics and student/instructor feedback.

ACTUAL OR ANTICIPATED OUTCOMES

This system was successfully implemented in MECH1750 in 2021 and enabled weekly Viva Voce assessments through peer marking in a cohort of 300+ students. Over 16,000 individual feedback forms were collected and distributed over the semester, with students able to give and receive feedback in real-time through a phone- and computer-compatible web interface. Both student and instructor perceptions of the system were positive. Instructors appreciated the reduced administrative workload that came with automatic collection and distribution, and students appreciated the timeliness and accessibility of feedback afforded by the system.

CONCLUSIONS/RECOMMENDATIONS/SUMMARY

In this paper, we present a purpose-built software system that facilitates scalable Viva Voce micro assessment through moderated peer marking. Given our positive experiences, we recommend the use of such a system in these classes more broadly and contribute a minimum working example at github.com/Cornelius2121/SVMS.

KEYWORDS

Purpose-Built Software; Industry Authentic Assessment; Viva Voce; Debate

Introduction

As universities aim to bridge the gap between theoretical education and industry practice, there is an increasing need for industry authentic assessments (IAA). One type of IAA is the Viva Voce (viva) – a form of oral examination – allowing students to verbally communicate their understanding and receive feedback on their discussion. Although viva assessments have significant benefits for student engagement/academic integrity (Sotiriadou, Logan, Daly, & Guest, 2020) (Ullah, 2020), and increase industry authenticity (Guzzomi, Male, & Miller, 2017), they are limited by scalability. Due to the time intensive nature of student presentations or discussions, and the real time assessment and evaluation, viva assessments are not typically conducted in engineering programs with large cohorts.

The teaching pedagogy designed to run a weekly debate-club style tutorial has been used successfully and has been described by Kirkland, Paardekooper, Flynn, Cuskelly, Prieto-Rodriguez, Mcbride and Gregg (2022). This work accompanies the pedagogy and describes the design, development, and deployment to facilitate such weekly debate-club style tutorials.

The structure of this tutorial style is contained in Kirkland, Paardekooper, Flynn, Cuskelly, Prieto-Rodriguez, Mcbride and Gregg (2022), however we provide a brief overview here. This tutorial method was utilised in MECH1750, a University of Newcastle undergraduate engineering materials course, with 300+ first year students. Students from a range of disciplines enroll in MECH1750, including mechanical, mechatronics, medical and aerospace systems engineering. In MECH1750, tutorial cohorts were distributed into groups of 4. In weekly tutorial, each group would participate in 4 debates against other groups. One student from each group would speak in each debate, with the remaining 6 students spectating and providing peer feedback for both speakers.

Following the completion of the tutorial, students were asked to rate the quality and helpfulness of the peer-provided feedback. The administrative workload of this peer review tutorial structure, if handled manually, would be unfeasible (~300 students $\times$ 4 debates = 1200 feedback forms to distribute each week).

Hence, a purpose-built web application was commissioned to automate the collection, sorting, distribution, rating and archiving of feedback - allowing for delivery of this class style to large cohorts in real time.

Design Requirements

Requirements and Features

The system was designed around 11 key requirements/features to facilitate scalable vivas. These entailed both student-facing and instructor-facing demands. Five core functionalities were scoped for student-facing features:

- S1: Spectating students can submit feedback on debates they witness.
- S2: Students can receive and ‘react’ to (action) peer-provided feedback on their own performance.
- S3: Students can revisit historical peer-provided feedback.
- S4: Students can compare their group performance standing against others.
- S5: Students can view their bracket matchup (group-group pairings, table location) for the upcoming tutorial.

Furthermore, 6 functionalities were scoped for instructor-facing features:

- I1: Instructors can submit feedback on debates they spectated.
- I2: Instructors can query and export bulk quantitative feedback for the purpose of marking.
- I3: Instructors can view unactioned feedback.
- I4: Instructors can view reported/flagged peer review feedback.

- I5: Instructors can view the full bracket matchup of viva micro assessments for each tutorial.
- I6: Instructors can view full standings of each group's viva micro assessment performance.

S1 & I1 - Submission of Peer-Provided Feedback

The submission of peer-provided feedback is an action required by both students and instructors. This functional requirement allows a spectator to submit feedback about two student participants in a debate. In accordance with the class design (Kirkland, Paardekooper, Flynn, Cuskelly, Prieto-Rodriguez, Mcbride and Gregg 2022), feedback is to be provided via a standard form with elements corresponding to key aspects of student performance. In this implementation, the metrics selected were: "Presentation Skills", "Technical Understanding", "Discussion" and "Professionalism". Each is assessable through a 4-point sliding scale ("Very Bad", "Bad", "Good", "Very Good") with a separate checkbox for "Uncertain" – where a spectator does not feel they have the knowledge to make an informed assessment of the metric. The use of a 4-point sliding scale inhibits spectators from fence-sitting with a "neutral" option. Two additional free-text fields, "Outstanding Features" and "Development Areas", allow spectators to make written comments.

S2 - Action Peer-Provided Feedback

In accordance with the class design (Kirkland, Paardekooper, Flynn, Cuskelly, Prieto-Rodriguez, Mcbride and Gregg 2022), students must be able to view, acknowledge and 'rate' the usefulness of peer-provided feedback. This final step 'closes the loop' and ensures that students process feedback each week. In our implementation, four potential 'ratings' were possible: "Thumbs Up - 👍" for helpful feedback, "Thumbs Nothing - ✊" for neutral feedback, "Thumbs Down - 👎" for unhelpful feedback, and "Flag - 🚩" for offensive/inappropriate/unprofessional comments. Rating the ~12 unique feedback comments provided each week must be a timely process. Flagged comments must be stored and easily queried for moderation by teaching staff (I4).

S3 - Revisit Historical Peer-Provided Feedback

For the purpose of tracking progress/improvement, a feature was implemented to allow students to revisit previous debate comments across all weeks for the purpose of tracking progress/improvement. Students should also be able to view their given reaction to this feedback.

S4 & I6 – Group Performance and Standing

In order to promote healthy competition, the average performance of their group compared to others in their tutorial/the wider class was presented to all groups. Likert responses ("Very Bad – "Very Good") must be represented quantitatively and both the rank and average score of each group should be visible to promote healthy competition.

S5 & I5 - Tutorial Bracket-Matchup

In each tutorial, pairs of competing teams debate. Pairs must be drawn in such a way to minimise repeat matchups. This requirement entails generating a suitable draw/bracket, and providing these pairings, and the debate location/table to teams.

I2 - Query and Export of Bulk Quantitative Feedback

Bulk quantitative feedback must be exportable for the purpose of grading and statistical analysis. A .csv file must be produced for easy consumption by teaching staff. Automatic calculation of grades based on staff-provided algorithms must be possible. Exporting a subset of data must be possible.

I3 - Unactioned Peer-Provided Feedback

For students who have provided feedback to have completed the evaluation of a participant, their feedback should be reviewed by the debate participant. To mitigate students not actioning peer-provided feedback, Instructors are to be able to view and action students unactioned peer-provided feedback for spectators to have their feedback evaluated.

I4 - Reported Peer-Provided Feedback

As discussed in functional requirement S2, students can flag offensive/inappropriate/unprofessional comments within peer-provided feedback. Instructors must be able to view inappropriate feedback to act externally to the system.

Interaction Medium

The system must be implemented in a medium that is easily accessible and has a short onboarding process for new users. Furthermore, the system must be scalable for hundreds of concurrent users. The solution was a dynamic web application.

The web application was to be hosted on a publicly accessible domain, for students to access the system remotely. To maintain system security, a login system should be implemented, with students provided accounts to access the systems functionality. The web application must also facilitate the instructor functional requirements and contain the relevant security measures to only allow instructors access to restricted functions. Furthermore, the web applications user interface must be designed to allow for robust functionality to be achieved both on a computer and mobile device web browser.

System Performance

For the designed system to support scalable vivas, performance requirements must be set to ensure the system is truly “scalable” for the 300+ students of MECH1750. These students must be able to simultaneously access the web application, submit and react to feedback, and view historical data. Furthermore, the system must support the query and export of bulk quantitative feedback, as per functional requirement I2. This process would be a demanding and computationally expensive task, that the system would need to process alongside managing users.

Implementation

The following section describes the implementation of a web application facilitating the above features and requirements. Given this implementation is purpose built for MECH1750, the software system is constrained to the exact features and requirements specified. The web application implemented was hosted on a public domain, for students to remotely access. Within the following section, a technology stack is defined, targeted at readers with a background in web application development. Furthermore, the implemented feature set is documented, outlining interface rationale and design.

Technology Stack

A commonly used web application technology stack was implemented to facilitate the functionality required. Firstly, a Python Flask (Ronacher, 2022) web application was developed to perform business logic, dynamically serve web page content, and manage database connections and queries. Utilising the embedded Jinja2 library within the Flask package, HTML web pages were “templatised” for the content of the pages to be dynamically served, but the style, layout, and functionality to be generic. Libraries such as jQuery, Bootstrap and Ajax were implemented to facilitate client-side functionality. Throughout the development of the web application, unit tests were developed to automatically test methods and full features.

Secondly, a PostgreSQL relational database was implemented to store data about students, debates, feedback, and other information related to the tutorial structure. The relational nature of the PostgreSQL implementation, enabled for complex single execution queries to be ran on the database, minimising the computation time of the business logic implemented in Python. The ER diagram of the database structure is shown in Figure 1.

Finally, the web application and database were deployed to an Amazon Web Services (AWS) EC2 instance (Amazon EC2, n.d.). An AWS EC2 instance is a cloud based virtual machine, that is typically utilised for web application deployment or computation tasks. In the deployment of the

Python Flask web application and PostgreSQL database, a dedicated EC2 instance, running an Amazon Linux 2 operating system, was setup as the endpoint for the applications domain. Nginx was utilised as an internal HTTP web server that was executed within the EC2 instance.

User Signup

For students to sign up to the web application, a “low-tech” solution was implemented to adhere to privacy concerns surrounding student users. To this end, student names were never collected or stored – only their university issued student number. To avoid password reuse with their university account, they were emailed a unique numerical pin code.

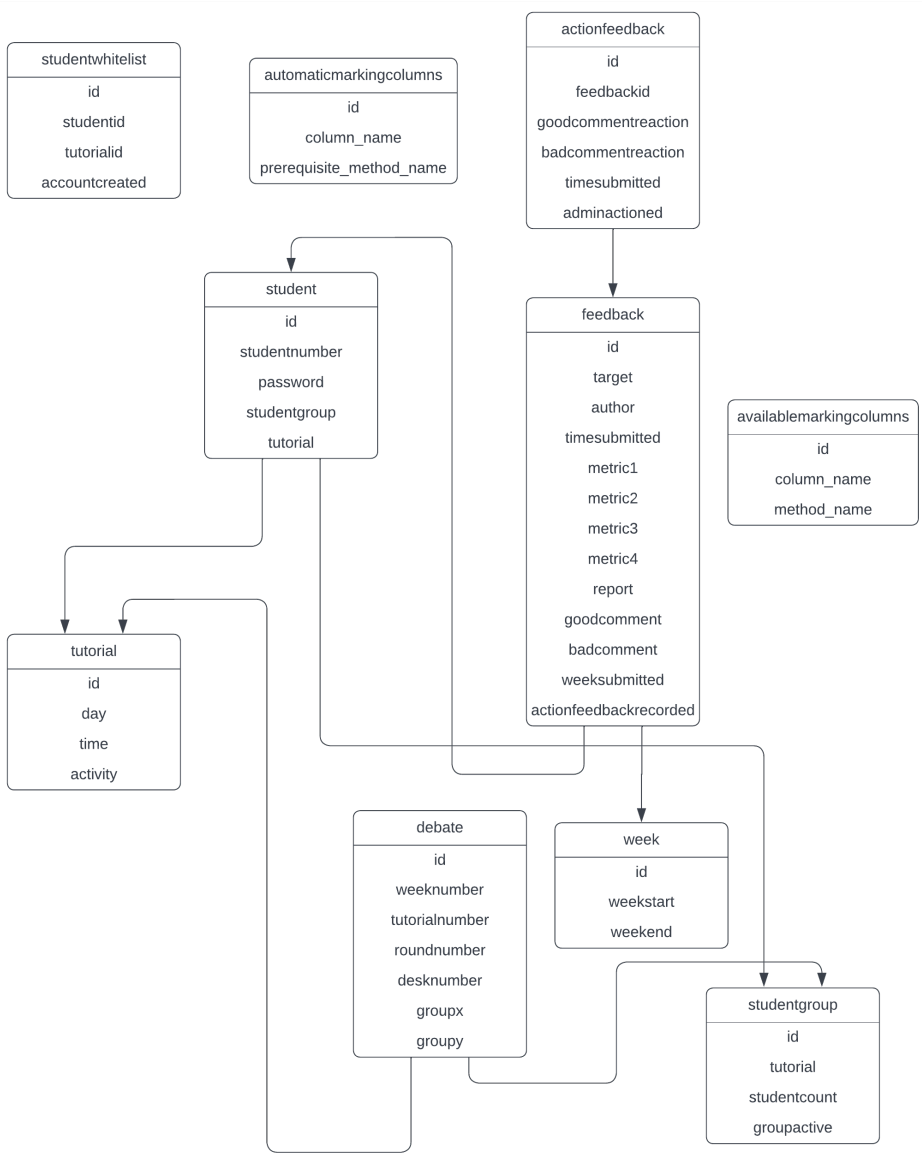

Figure 1: ER Diagram of Database Structure.

Group Registration

A registration system utilising a web form within the web application was implemented. Once four students had individually signed up to the web application, one student could register a debate group through a portal on the web application. This simultaneous signup precluded the need for a pairing/mapping system for asynchronous registrations and ensured that no under- or over-sized

groups could be formed without instructor intervention. Figure 2 shows the implemented group signup form.

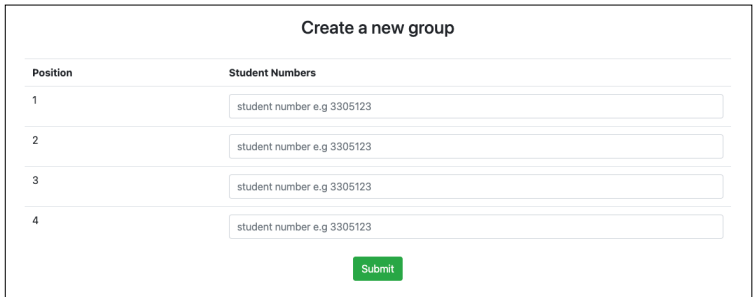

Figure 2: Group Signup Form.

Feedback Submission and Review

A two-part feedback submission form was developed. In each part, feedback of a debate participant could be entered as shown in Figure 3. A sliding point scale was implemented to input students' performance metrics. Both the values of the sliding point scale and the metrics themselves can be easily renamed and customised. Warnings were shown if the target of a feedback form was not scheduled to debate the author of that feedback. This warning could be overridden, to allow for edge cases, e.g., an entire group being absent. Upon receiving feedback, students could rate the quality of the feedback provided through a dedicated review page, as shown in Figure 4.

Tutorial Bracket Generation and Display

A system was developed to facilitate the generation of debate brackets/matchup/schedules. This problem is nontrivial; for a debate bracket/matchup/schedule to be valid, groups must:

1. Not be scheduled to debate themselves.
2. Not be scheduled to debate the same team twice in a single tutorial.

A bracket should also minimize the number of repeat matchups across consecutive tutorials. These principles must be adhered to over all four pairings in a single tutorial. Arriving at such a derangement was relatively straightforward with a regimented strategy: a static list and rotating deque data structure were implemented. Unique group identifiers were stored within both the list and deque, and for each matchup, the deque was rotated by one position, to create a new, non-duplicate matchup. This rotation was executed 4 times per tutorial, to generate the full tutorial bracket for a given week. The state of the deque was saved and used for the generation of each tutorial, to minimize the number of repeat matchups across consecutive tutorials. The tutorial bracket was displayed to the user in table format, showing only their groups bracket matchup for the current week.

Administrative Tools

Feedback moderation allowed instructors to view student feedback comments that have been reported, through the flag 'rating'. The reported comments were displayed tabularly, showing the unique identifier of both the student who submitted the comment, and the student who reported the comment. This allowed instructors to act externally to the system.

Instructors could also provide feedback, utilising the same form as students.

Additionally, in the case where students had not reviewed feedback provided, instructors could quickly action feedback that had not been actioned at the end of a given tutorial period. Finally, an automated grading function was developed to mux quantitative performance statistics and minimum-requirement actions to generate grades for all students from a given weeks tutorial.

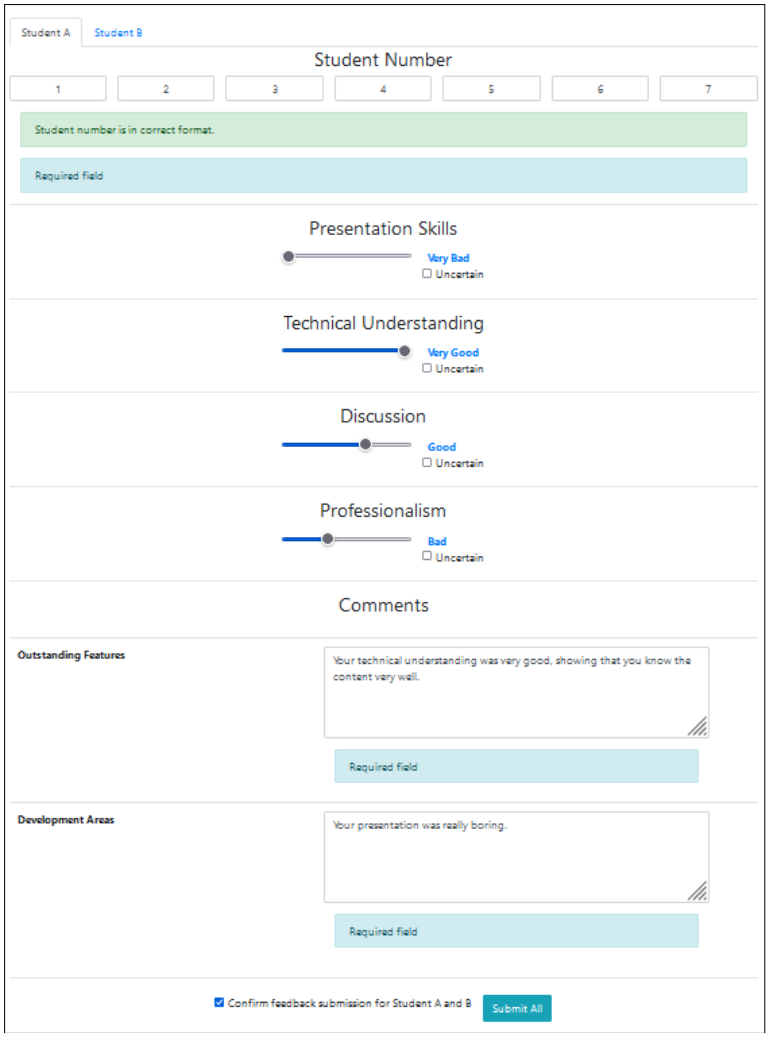

Figure 3: Web Application Feedback Submission Form.

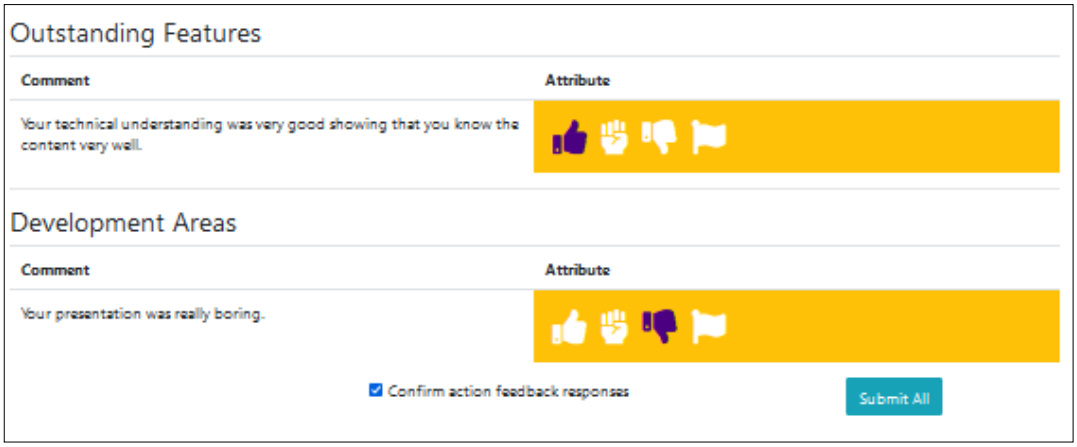

Figure 4: Web Application Feedback Review Form.

Evaluation

The system performed to the required standard for the web application to be functional, met the required maximum user load and maintained appropriate response times. Over the course of the semester, 2,000 debates took place, involving 300+ participants from 75+ teams. Over 12,000 individual pieces of feedback were lodged. Within each tutorial, typically 17 simultaneous debates were conducted with over 140 active participants.

When feedback was submitted by students during tutorials, the average response time was less than 500ms, aligning the web application and server configuration with acceptable user standards.

The quality and performance of the web application was also indirectly visible through student feedback and comments. 102 students participated in the end-of-course evaluation survey, where students had the opportunity to provide a course quality of learning experience (QLE) score and leave written feedback statements. The average QLE score for MECH1750 was 4.92 out of 5, indicating students responded positively to the course and debate tutorial style. Within the written feedback statements provided, there was zero negative comments made about the online feedback platform from any student across all student feedback. This further highlights the positive and effective performance of the web application in facilitating the tutorial style.

Instructors received the implemented system very positively. Through a review of the web application and its deployment, instructors noted 2 areas of the system that were valuable for wider peer-review assessments, the tutorial bracket-matchup and feedback submission forms features. Firstly, the tutorial bracket-matchup feature aided instructors in running the class more effectively, as traditionally manual group bracket-matchups can be a time intensive task. Secondly, instructors commented that the feedback submission functionality effectively implemented the designed teaching pedagogy (Kirkland, Paardekooper, Flynn, Cuskelly, Prieto-Rodriguez, Mcbride and Gregg 2022) for MECH1750.

To further evaluate the performance of our solution, we compare functionality with two paid, “off-the-shelf" solutions: FeedbackFruits and Buddycheck. In the FeedbackFruits platform, the developers create a strong emphasis on the benefits of learning management system (LMS) integration, with common LMS’ such as Canvas and Blackboard. This is also a common feature of the Buddycheck platform. However, both FeedbackFruits and Buddycheck require reoccurring evaluation setup by instructors, as opposed to the proposed systems automated scheduling of feedback collection. Furthermore, the free and open-source nature of the proposed system allows for it to be competitive with paid “off-the-shelf” solutions, even without LMS integrations.

Web Application Deployment

This section outlines the high-level procedure for the deployment of the software implementation and is targeted at readers with a background in web application development. An anonymised version of the web application source code has been made available as an open-source repository at github.com/Cornelius2121/SVMS. For the web application to be deployed as a new instance, the following information is required:

- Each tutorials weekday, start time and activity number,
- Each student’s unique identifier,
- A mapping of students to their enrolled tutorial, and
- A list of weeks that each of the tutorials will run.

Furthermore, for the web application to be made available to users, the following infrastructure is required:

- Internet domain,
- Infrastructure level application server, such as an Amazon EC2 instance, and
- Email address associated with the internet domain.

To deploy the web application, the following procedure should be followed:

1. Source code should be deployed to the application server.
2. PostgreSQL instance should be created, initial required data inserted, and configuration information added to the web server configuration script.
3. Configure Nginx to direct HTTP traffic to the web application.
4. Configure the domain to direct traffic to the IP address of the application server.

To contribute to the open-source repository, a pull request can be submitted on the repositories page at github.com/Cornelius2121/SVMS.

Conclusions and Future Work

In this paper, we outlined the design, functionality implementation and deployment guidelines for a purpose-built software system to facilitate large scale viva micro assessments. These viva micro assessments were run in the form of debate-club style tutorial sessions. The software system was required to schedule debate sessions, facilitate user, and group signup, collect peer-provided feedback from each of the debate sessions, allow for peer-provided feedback to be reviewed and 'rated', and provide a range of administration features.

The purpose-built software solution designed to fulfil the system requirements was developed as a Python Flask web application and deployed on an Amazon EC2 server instance. The system was successfully implemented in 2021, in the University of Newcastle course, MECH1750. The system performed effectively, meeting all system design requirements. Furthermore, the system maintained acceptable industry standard response times, will facilitating over 2,000 debates, and collecting more than 12,000 pieces of peer-provided feedback across the duration of a semester.

The source code for the purpose-built software system has been made open-sourced and is available at github.com/Cornelius2121/SVMS. In future development of the software system, various features will be introduced to generify the system, to allow for a similar tutorial style to occur in other disciplines apart from Engineering. Furthermore, as the system functionality includes the moderation of peer-provided feedback, the web application could be expanded to function as a tool to facilitate a variety of peer-review assessments. This work entails additional configuration items, customisable feedback forms and modularisation. Similarly, to aid deployment, we aim to provide a Docker image for the web application and database, reducing the barrier to system deployment. Finally, we aim to develop further customisation to our automated grading function, allowing for new functions to be utilised for individual students.

References

Amazon EC2. (n.d.). Retrieved July 12, 2022, from https://aws.amazon.com/ec2/

Guzzomi, A. L., Male, S. A., & Miller, K. (2017). Students' responses to authentic assessment designed to develop commitment to performing at their best. *European Journal of Engineering Education*, *42*(3), 219–240. doi:10.1080/03043797.2015.1121465

Kirkland, A., Paardekooper, C., Flynn, J., Cuskelly, D. Prieto-Rodriguez, E., Mcbride, B. & Gregg, A. (2022). *Scalable Vivas: Industry Authentic Assessment to Large Cohorts through "Debate Club" Tutorials*. Paper presented at the Australasian Association for Engineering Education Annual Conference, Sydney, NSW.

Ronacher, A. (2022). Flask Documentation. Flask Web Development. Retrieved July 12, 2022, from https://flask.palletsprojects.com/en/2.1.x/

Sotiriadou, P., Logan, D., Daly, A., & Guest, R. (2020). The role of authentic assessment to preserve academic integrity and promote skill development and employability. *Studies in Higher Education*, *45*(11), 2132–2148. doi:10.1080/03075079.2019.1582015

Ullah, S. N. (2020). Examples of Authentic Assessments in Engineering Education. *2020 IEEE Global Engineering Education Conference (EDUCON)*, 894–897. doi:10.1109/EDUCON45650.2020.9125271

Copyright statement

Copyright © 2022 Paardekooper, Flynn, Kirkland, Cuskelly, Gregg: The authors assign to the Australasian Association for Engineering Education (AAEE) and educational non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The authors also grant a non-exclusive licence to AAEE to publish this document in full on the World Wide Web (prime sites and mirrors), on Memory Sticks, and in printed form within the AAEE 2022 proceedings. Any other usage is prohibited without the express permission of the authors.