EPiC Series in Computing

Volume 69, 2020, Pages 383–392

Proceedings of 35th International Conference on Computers and Their Applications

A Scalable Cloud Native Platform For Interactive Museum Exhibits

Paul Cappaert and Alex Redei

Department of Computer Science, Central Michigan University
Mount Pleasant, Michigan
(cappa1pt, redei1a)@cmich.edu

Abstract

Science museums with hands-on and interactive exhibits are on the rise. As museums grow, the need arises to have an online platform to allow visitors to continue their experience beyond a day visit. In this paper, we first provide a brief survey of techniques for building scalable cloud native software frameworks. In order to achieve low cost and persist user data, we built a django application using Heroku and postgres. This platform can be scaled horizontally on-demand to handle highly variant user traffic for augmenting the museum experience. With a focus on educational experiences, a participant's progress on activities at the museum are saved through an API we built and can be viewed on a website. Different activities at the museum generate data for the API which can be viewed anywhere. Simulated data was loaded into our framework to validate the efficacy of our solution. Future testing is outlined in collaboration with the Fleischmann Planetarium through a trial experiment with museum visitors.

1 Introduction

The museum of science and industry is a world-renowned interactive museum attracting over 1.5 million attendees every year. The museum provides interactive exhibits to get students involved and excited about science and learning. [3] One such exhibit is shown in Figure 1, where a Baxter robot was programmed to play tic-tac-toe with two students.

A shortcoming of even the best museum experience is that the learning ends when you leave. We believe that there would be a lot of value added if the activities you participated in the museum could be recorded and made available beyond the walls of the museum. This is the primary aim of this project - to enable museum visitors to look back at the results of their experience. According to Dr. Sharma, "Reflections are what psychologists call an 'episodic grasp of reality', which are not only experiences but also excellent learning opportunities" [9]. An opportunity to reflect on museum activities strengthens learning outcomes. [7]

The secondary aim is to create a platform where you could revisit the same museum and continue where you left off in a specific activity. [edit] This would allow for a more in-depth educational experience. A further benefit of such a platform would be the ability to continue the same experience at a different museum.

Figure 1: (left) A Baxter robot in use at the Chicago Museum of Science and Industry [6] (right) A picture of the final station of the be the astronaut exhibit [12]

The inspiration for this project is the Be The Astronaut program at the Fleischmann Planetarium. In that program, students participate in a variety of educational games geared around different astronaut activities. Students are given a personalized astronaut ID card [1]. Students use the card to sign in to the different stages in the exhibit which allows the museum to track their progress. The ID card system is limited to the activities in the Be The Astronaut exhibit. You can see a picture of the third activity in Figure 1.

The system we are creating is to augment a museum experience such as the Be The Astronaut exhibit. Our platform can track results from different activities and save them to a database where they can be viewed from our website. This system is designed to be portable between different museums. For the same game the same data will be saved for every user and every user could use each site interchangeably.

To the best of our knowledge there is no system that fills this educational niche for museum exhibits. A verbal survey of two museum directors was conducted at the beginning of this project to determine what other solutions might be available. As of the time of this writing we are unaware of other software that provides this feature set.

The rest of this paper is organized as follows: Section 2 is devoted to the background and related work of other applications of cloud tools, system design is presented in Section 3, software architecture and methods of this solution is presented in Section 4, Section 5 address the the user experience and contains screenshots of our prototype in action, and finally Section 6 wraps up our findings with several concluding remarks and directions for future work.

2 Background

When developing software to be deployed to the cloud there are in general two types of services that can be used. IaaS (Infrastructure as a Service) and PaaS (Platform as a Service). According to Sun, Wu and Huang, IaaS "provides hardware equipment and other infrastructure services, and makes the basis of centralized computing resources (CPU, memory, network) can work independently." PaaS is an extension of IaaS. In addition to the hardware, "it provides the software deployment runtime environment which is based on the infrastructure as a service. Platform as a service will be responsible for the entire life cycle of software. It includes the deployment, running and stops, and at the same time it monitors the loads of software systems, fault tolerance, safety and other running time state." [11].

A crucial question is which type of service to use. PaaS has the potential to save a lot of time in configuration and deployment. According to Sharma and Santharam, "PaaS brings with it the promise of a convenient environment to develop and deploy applications, while not having to worry about the underlying infrastructure or operating systems."

However, PaaS is not always appropriate. "Typical use of PaaS platform has mostly been for lightweight stateless services and applications (like portals), rather than for complex multi-tier applications." [10]. In general, when fine grain control is needed for a service, IaaS is a better choice. If you are developing a more typical application, such as a stateless web app, PaaS is the better option.

The choice of what framework to use is crucial to building an application. What is probably the most important consideration is that a framework must be mature and have a large and robust community to support it. Django is a pure python web framework with a large community and many built in features. According to Liawatimena et all, "Django is also scalable, mature, and fast with a numerous community of developer and a robust set of built-in components. Django can access or create JSON or XML data instances and works out-of-the-box with relational database management systems such as Oracle, MySQL, SQLite, and PostgreSQL. At deployment, Django is fully backed up by the cloud platform AWS Elastic Beanstalk and Heroku." [5]. Because of these advantages Django development is fast and flexible. Many features, such as database support, come built-in with Django. Because of Django's popularity has a large ecosystem of supporting services. Deploying Django to a PaaS service such as Heroku is simplified. Steps such as building and containzering are done automatically by the platforming allowing developers to focus more on adding features.

Our system will be completely cloud native. We are utilizing Heroku, which offers PaaS - Platform as a Service. Paas is ideal for rapid application development as it removes the need to build one's own infrastructure [8]. Heroku provides our web and data hosting in a seamlessly integrated system. For scaling, Heroku also offers tools for controlling the specific containerization of our application. For the development phase of our project the free tier of Heroku is sufficient, and we can start scaling when needed.

Heroku offers cost effective scaling for our purposes. We anticipate that in a production environment our load will be highly variant, with the majority of time very low load and periods of heavy use. For example, our museum activity will have 100 students during peak hours, and little to no use in the opening hours. Traffic at other times will likely be very low. Because of this environment we will require flexible scaling where we can ramp up the power of our service at peak times. Adjusting the scale as such will significantly reduce our cloud computing costs.

The way we deploy to Heroku is by containerizing our web application. Horizontal scaling can be done by adjusting the number of containers, called dynos, running our app. Heroku prorates the usage of dynos to the second so we only have to pay for what we need. Dynos are very lightweight. Spinning up additional dynos appears to happen quickly and only takes a few seconds. Dynos only contain the dependencies that our application needs. To be prepared to manage higher traffic, we could tell Heroku to spin up additional dynos as the system comes under load. From the perspective of the user, the performance will thus remain consistent. This information was procured from the Heroku documentation [2].

3 System Design

For the development of this project, we used the free version of Heroku. This gives us access to a low power dyno. In Figure 2 below you can see the control panel with our application

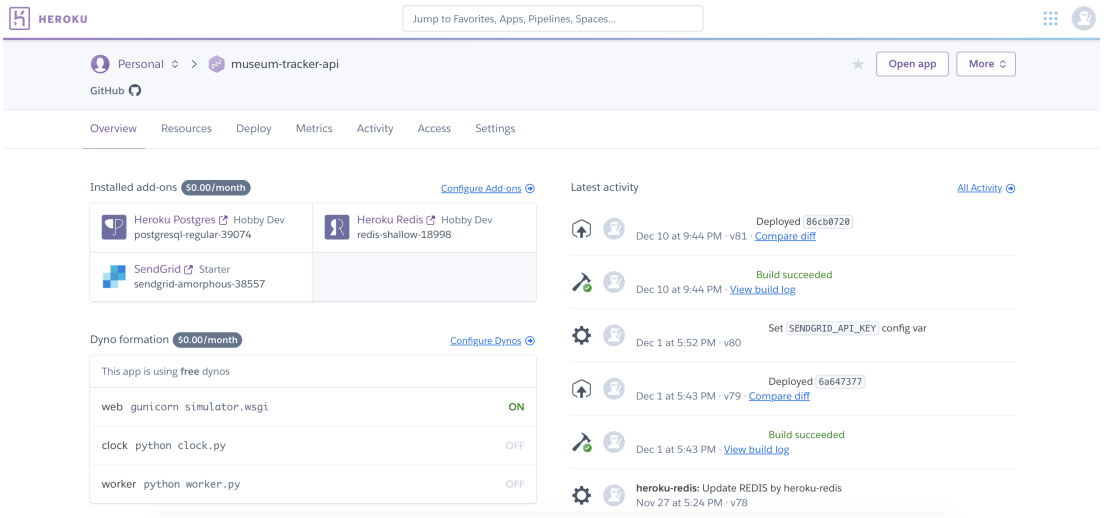

Figure 2: The Heroku control panel with our deployed solution. Sensitive information was cropped from this screenshot

deployed into a dyno on the platform. From this control panel we can manage deployments, add on new services, and view our current running dynos. Heroku simplifies all of these processes so that we can focus our attention on developing the platform.

The museum activity being tracked in our prototype is a flight simulator. The data created by the simulator about the user is sent to the cloud via http. An overview of the systems's design can be seen in Figure 3.

The museum tracker API connects to the server locally using a Raspberry PI or a micro computer. The PI will then read UDP input from the simulator and create an HTTP request to send to the server. If there is some kind of error in sending the request it will be saved to an microSD card so that it can be sent later when the error is resolved. Such an error could be caused by a network failure on premise, unexpected outages on the server, or any other of a number of common issues with running a distributed system.

The PI is also connected to a handheld usb QR barcode scanner via bluetooth. You can see the scanner and PI in Figure 4. The program running on the PI will listen for the data from the scanner and send that along with the data from the simulator in a REST call. Using a QR code to identify allows for a high amount of reliability. The data encoded in the QR code has error correcting features which allow part of it to be partially obscured or corrupted and it can still be read correctly, as demonstrated by Dr. Li, et. all's paper on aesthetic QR code generation with masking. [4]

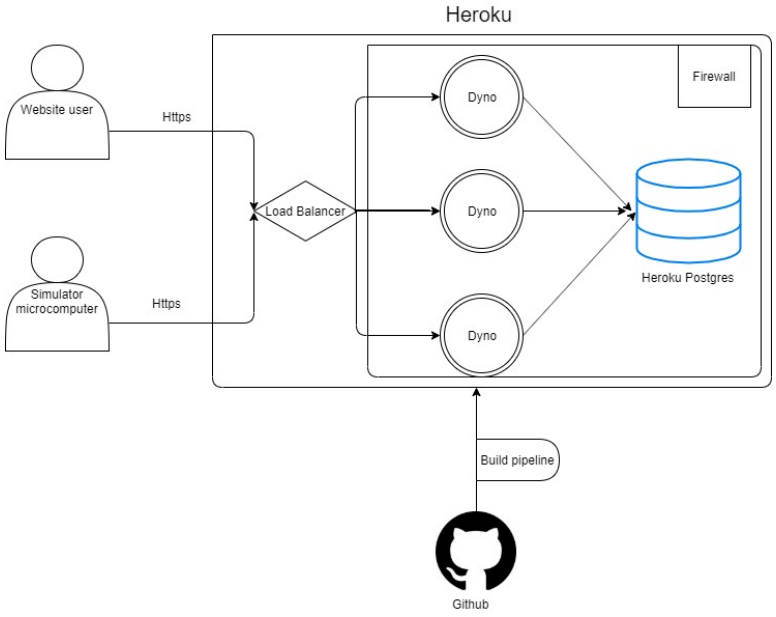

Figure 3: A high level system design of the Museum Tracker API

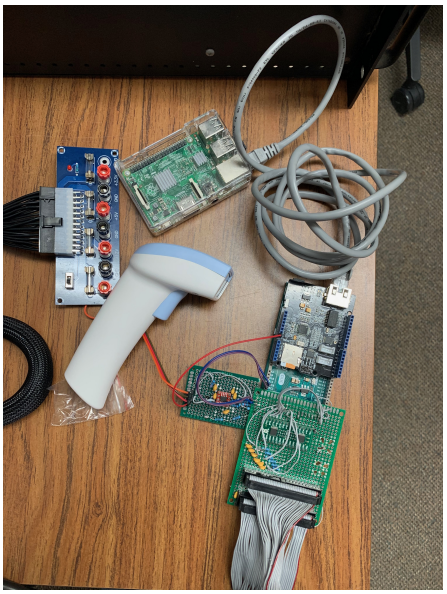

Figure 4: QR code scanner and Raspberry PI

4 Software

Django is the python framework we have used to build our web application. The Django framework was chosen as it contains many features expected of a web application, such as

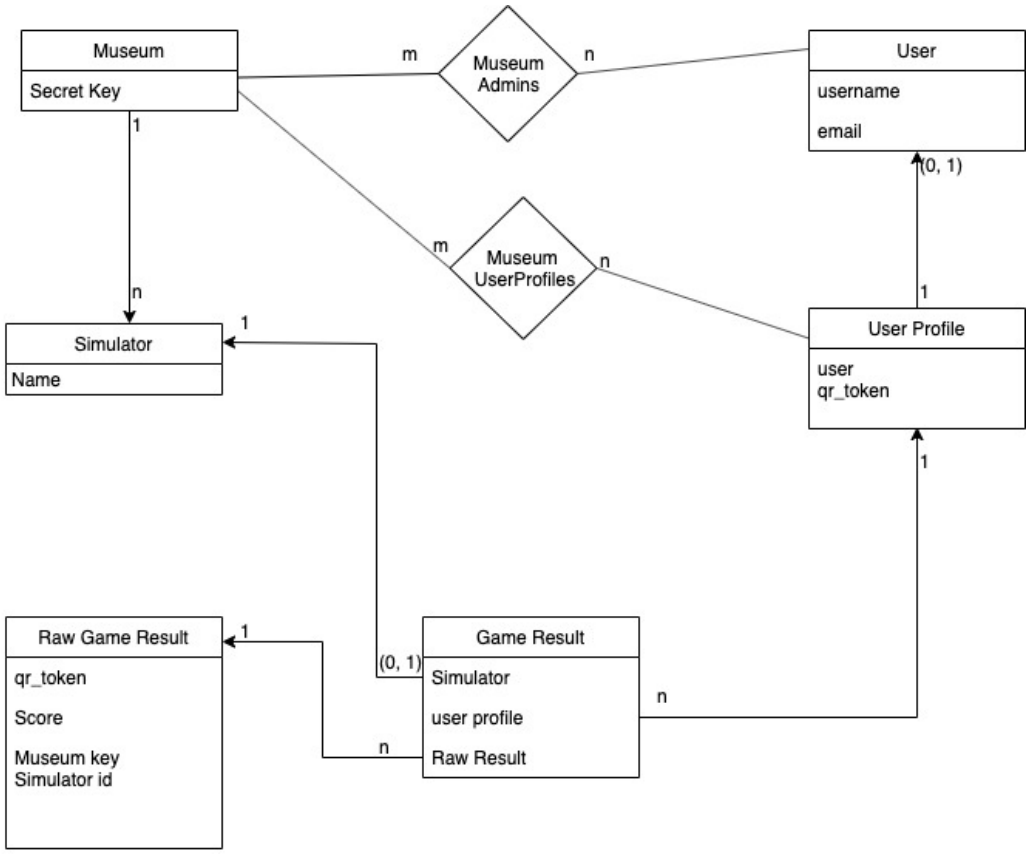

Figure 5: An ER Diagram of the Museum Tracker Database

secure user registration. This allowed us to get a working prototype in front of our stakeholders relatively quickly. Using Django also gives us access python libraries. Python has many easy to use libraries to help us complete tasks such as pdf generation and qr code rendering which are also used in our application.

The database for our application is using Postgres. Postgres is a proven SQL database system and easily integrates with Django. The Django ORM - Object Relational mapping - allows for easy persistence of logical models without needing to create a database layer in the application. With the migration function, the Django ORM takes the Python classes and creates the necessary SQL for the models to persist to the database. The design of our database can be seen in Figure 5.

Whenever a game result is sent to the API correctly a Raw Game Result object will be saved to the database. This will then trigger a function which will check if all of the features of the raw game result are correct, and if so it will create a game result object which references the raw game result, the profile, and the simulator objects. The reason for this extra layer of abstraction is that if there is incorrect information that result will still be saved for later diagnosis. For instance, if the museum admin incorrectly puts in the simulator ID but all of the other information is valid they will be able to fix this later and the users scores can still be saved on the website.

The userprofile object contains the QR token to be used to interact with the simulators. The user object is created after the userprofile object when the user registers on the website. This design decision would allow a participant to get a qr code from the site admin and start using the simulators right away without having to register an account. They would be able to register at any point in the future by following the link on the qr code.

5 User Experience

There are two types of users for our system - the museum visitors and the site admin. The site admin would be the person directly responsible for the museum activities. In our solution, they have the ability to change information for their particular museum, but not others. They also have access to tools to help make the setup of the back-end data connection as simple as possible.

In our system, the site admin needs to be able to easily manage the experience for their simulators. They will need to be able to...

1. Setup the connection from the interactive museum activities to the API
2. Register multiple users at once and print out QR codes for those users
3. Be able to manage those users
4. Keep track of and display results of the activities, like high scores

The site admin has access to a control panel, for their museum where they can get a pdf of QR codes to be used by the participants. They can look at a management page for their museum where they can view user activity and can hide users and flag them for removal from an admin - if for instance, a user registers an inappropriate name. The site admin will only have access to these controls for their own museum. You can see this page populated with test data in Figure 6. Usernames can be one of three things: alphanumeric, e-mail addresses, or unregistered. Unregistered means that a profile was created by the museum without the website user being registered for that profile. In Figure 6, we edited out the e-mail addresses for this paper.

The process will begin with students receiving a QR code from the museum admin. The museum admin will also be able to generate up to a hundred QR codes at once on a pdf. You can see an example of QR codes generated in our system and printed out in Figure 7. In a museum setting, these codes would be printed on special label-making paper and then added to museum ID cards. The QR codes are then used for interacting with the museum games and also for registering an account on the website. An additional benefit to this approach over an RFID identification system is that you can easily register an account by using the built-in QR code reading function of your smartphone.

The students will use the QR code at educational games in the museum and afterwards the results will be saved for them on the website. The results can be viewed on an individual profile page or on an overall museum leaderboard, as demonstrated in Figure 8. Access to the leaderboard will be under the control of the museum admin for privacy protection.

Museum-goers can register for an account only after they've visited the museum and received their QR code. Registration can happen either using a smartphone or a computer with QR reading capabilities. They will then be able to view the results of their museum visit online. The user's name and email is not required for QR code registration, thus allowing the museum to generate large amounts of trackable IDs without slowing down the reception counter at the

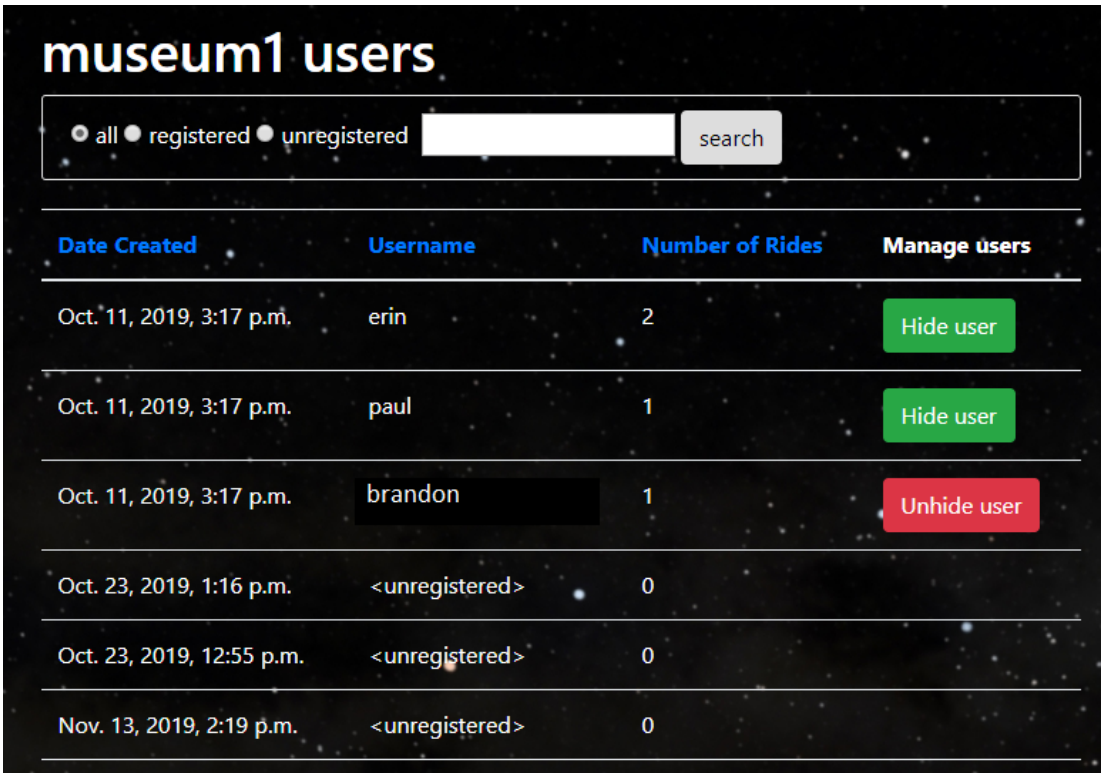

Figure 6: Museum admin user control panel

museum with user sign-up. Registration can happen in the museum at a later time, or after the museum experience. Another feature available to the museum admin is tha ability to show participants a leaderboard page where they can see how they've scored in comparison to others. The creates a novel competitive aspect to an otherwise one time experience. You can see an example of the leaderboard in Figure 8.

Figure 7: QR codes for user registration

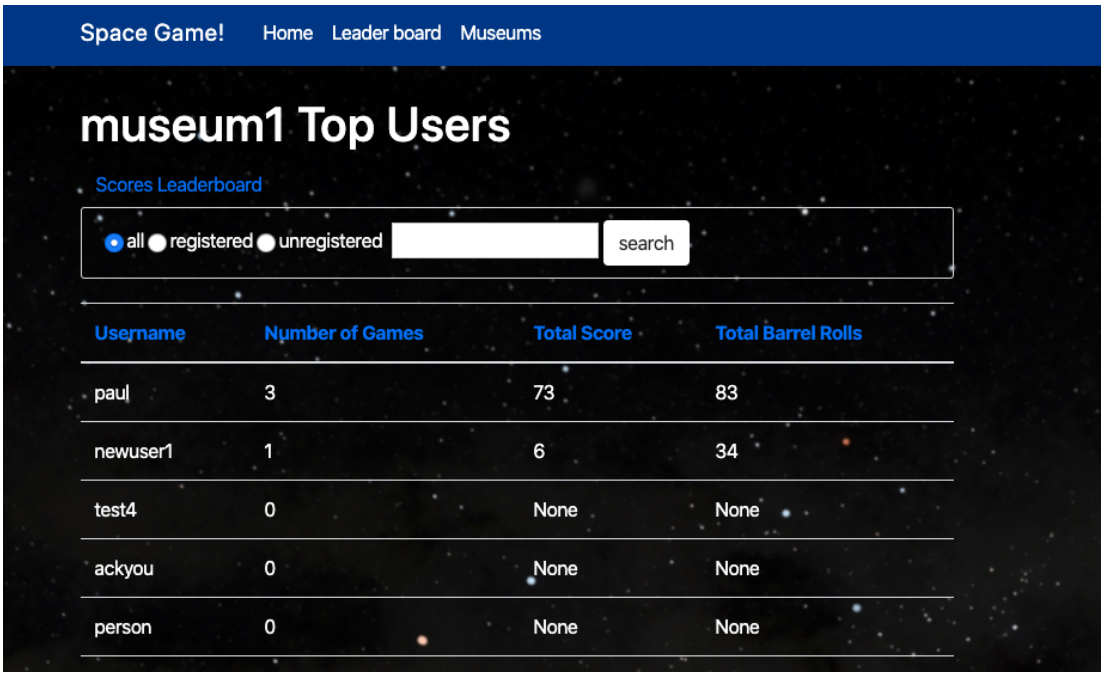

Figure 8: Museum leaderboard

6 Future Work and Conclusion

In this paper, we presented a platform for museum activity tracking. The initial prototype is a cloud-native solution that showed that our API can scale horizontally as the demands on the system changes. We demonstrated a working API populated with simulated data, our user interface, and tracking of museum participants from generated QR codes.

For future work, we are planning to test our system with real museum attendees. We are working with Paul Mcfarlane, the director of the Fleischmann Planetarium, to have a group of 30 students try our system in December. This will help us test the reliability of the system we have built beyond the simulated tests. We will get real world data about our scaling needs and the ability of the framework to meet those needs. For that reason, we are planning to stress test the system with students simultaneously attempting to register and use the system at the same time. This test will also provide feedback our user experience and graphical interface, enabling us to gain an understanding of what the experience is like for participants.

References

[1] San diego air space museum. *San Diego Business Journal*, 37(44):1, Oct 2016. Copyright - Copyright San Diego Business Journal Oct 31-Nov 6, 2016; Last updated - 2016-11-28.

[2] Salesforce Inc. Heroku documentation, Dec 2019.

[3] Steve Johnson. Chicago museums set attendance records in 2016. *Chicago Tribune*, Jan 2017.

[4] Li Li, Jinxia Qiu, Jianfeng Lu, and Chin-Chen Chang. An aesthetic qr code solution based on error correction mechanism. *Journal of Systems and Software*, 116:85–94, 2016.

[5] S. Liawatimena, H. L. Hendric Spits Warnars, A. Trisetyarso, E. Abdurahman, B. Soewito, A. Wibowo, F. L. Gaol, and B. S. Abbas. Django web framework software metrics measurement using radon and pylint. In *2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*, pages 218–222, Sep. 2018.

[6] Melissa Merli. Robots invade chicago's museum of science and industry. May 2017.

[7] Alexander Redei and Sergiu Dascalu. An educational science ride using a motion flight simulator platform. *Procedia Computer Science*, 126(1):1666–1672, Aug 2018.

[8] Alexander Redei, Sergiu Dascalu, and Frederick C. Jr. Harris. A framework for virtualizing joystick controls in a flight simulator training environment. *International Journal for Computers and Their Applications*, 26(1), Mar 2019.

[9] Monika Sharma. Learning through reflections. *CHRISMED Journal of Health and Research*, 5:319, 01 2018.

[10] V. S. Sharma and A. Santharam. Implementing a resilient application architecture for state management on a paas cloud. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 1, pages 142–147, Dec 2013.

[11] F. Sun, Q. Wu, J. Huang, and Y. Yu. An universal automatic configuration software model based on the cloud computing. In *2013 5th IEEE International Conference on Broadband Network Multimedia Technology*, pages 306–311, Nov 2013.

[12] Wings Over the Rockies Air Space Museum. Be the astronaugt exhibit, Dec 2019.