EPiC Series in Computing
Volume 64, 2019, Pages 215–224
Proceedings of 28th International Conference on Software Engineering and Data Engineering

ARIA 3.0: A Modern Approach to Web-based Music Festival Registration Systems

Nikkolas J. Irwin, Anthony Bennett, Kevin Carlos, Jalal Kiswani, Cynthia R. Harris, Sergiu M. Dascalu, Frederick C Harris, Jr.

Department of Computer Science and Engineering,
University of Nevada, Reno, NV USA
fred.harris@cse.unr.edu

Abstract

Administration, Registration, and Information Assistant (ARIA) 3.0 is a full-stack web-application developed to assist the Northern Nevada Music Teachers Association (NNMTA) with numerous music festivals and competitions occurring annually. ARIA's primary functionalities are the following: event creation, event scheduling, event management, user management, document processing, and payment processing. Prior to the creation of ARIA, these functionalities were performed using-paper methods. This approach was tedious, inefficient, and prone to errors. The previous versions of ARIA (1.0/2.0) were developed to address these issues using the WordPress platform and WordPress plugins. While successful, new business requirements proposed by NNMTA have demonstrated that WordPress may not be a feasible software platform for all of NNMTA's needs going forward. To ensure that ARIA can advance with NNMTA, ARIA 3.0 was developed using an "as-is" and "to-be" design approach. We built a microservice application which supports the implementation of features that NNMTA would like to maintain from previous versions while simultaneously providing a platform suitable for future software requirements. Additionally, new portals built for customers, students, teachers, and administrators aim to improve the user experience (UX) and user interface (UI) by increasing engagement with ARIA through new functionalities, greater access, and greater control over information.

keywords: event scheduling, event management, document processing, festival, full-stack, microservices, microservice application, microservice architecture, music, NNMTA, payment processing, portals, registration, web-application

1 Introduction

Administration, Registration, and Information Assistant (ARIA) 3.0 is a full-stack web-application currently in development which builds upon its predecessors (1.0/2.0) [8] using modern web tools and technologies to address the changing business requirements of the Northern Nevada Music Teachers Association (NNMTA) and remove constraints innate to the WordPress platform and WordPress plugins. ARIA 3.0 has been in development by its authors since

2018 and it is continuing to be developed further to implement additional services and features with the ultimate aim of replacing the current version (ARIA 2.0) [8].

ARIA 3.0 is being sponsored by NNMTA, a music organization in Reno, Nevada with the purpose of aiding NNMTA with music festivals and competitions. In these events, students whose musical expertise varies, perform in front of judge(s) and receive feedback regarding their performance. The nature of these performance varies in duration from a few minutes to around an hour. Further information regarding the origin of NNMTA, its parent organizations, and comparisons to other event management software can be found in Section 3 of "ARIA: Efficient Music Festival Manager" [8].

Before ARIA's introduction, NNMTA had to perform all event scheduling and event management tasks by hand. This approach was tedious, inefficient, and error prone. Once ARIA was introduced, the overhead encountered by NNMTA staff was significantly reduced [8]. Many affairs previously conducted by hand were now managed electronically via WordPress and WordPress plugins [8]. The introduction of ARIA also improved the experience for customers and teachers by automating most of their involvement with NNMTA.

Previous versions of ARIA (1.0/2.0) demonstrated the beneficial role that software could perform for NNMTA [8]. While a significant improvement from paper methods, some current and future business requirements have shown that ARIA needed to be updated so that it could develop new features without potential WordPress platform and WordPress plugin constraints. ARIA 3.0 attempts to resolve these concerns by providing NNMTA with a microservice application equipped to maintain the majority of existing features from ARIA 2.0 [8] while also providing an ideal platform for future development.

The rest of this paper describes the following: Section 2 covers the motivation for creating ARIA 3.0 and its design, Section 3 provides an overview of our work and highlights some major features and services currently implemented in ARIA 3.0, and Section 4 describes observations about our work and features to be added to the application.

2 Motivation and Design

The technical specifications for ARIA 3.0 were developed with Mrs. Cynthia Harris, Chairwoman of NNMTA music festivals, Dr. Frederick C. Harris, Jr., computer science and engineering faculty at the University of Nevada Reno (UNR), and our technical advisor, and Dr. Jalal Kiswani, a PhD candidate at UNR at the time of ARIA's development.

The authors of ARIA 3.0 typically met bi-weekly to discuss the requirements, design, implementation, integration, and operation of the application. The meetings were an essential component of ARIA 3.0's development and assisted the team by identifying strengths and weaknesses of the previous versions (1.0/2.0), determining which aspects of previous versions our stakeholders wished to maintain in the new application, and prioritizing new features for ARIA 3.0. Our team implemented this feedback by applying a hybrid of feature driven development (FDD) and scrum software development methodologies where we focused on feature-migration, a living backlog of prioritized work to be done, and short iterative development cycles.

The major goals of ARIA 3.0 were the following: create a platform to support the future business requirements of NNMTA without WordPress platform constraints, further reduce the overhead of NNMTA, improve ARIA's UX/UI, and expand the application so that customers and teachers have greater access to information via their own portals. These goals are documented through the software specifications listed in Subsections 2.1, 2.2, and 2.3.

2.1 Functional Requirements

The following functional requirements are "statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do" [10].

1. The system shall provide administrators a form for event creation.
2. The system shall provide customers a form for registering a participant (student).
3. The system shall provide a form to facilitate account creation.
4. The system shall provide administrators a form to schedule the music festival event.
5. The system shall provide administrators a form to schedule the command performance.
6. The system shall provide a portal for customers.
7. The system shall provide a portal for teachers.
8. The system shall provide a portal for administrators.
9. The system shall provide document generation. These documents shall be printable, downloadable, and easily distributed through printed and electronic methods.
10. The system shall use a variety of payment processing methods (e.g. PayPal) to provide customers with flexible payment options.
11. Successfully created events will be made public and propagated to portals.
12. Customers who successfully register a participant (student) using the event-scheduler will be redirected to PayPal to complete registration payment.
13. After a payment is confirmed, participants (students) will be redirected to the customer-portal's view active registrations page where their new registration will be displayed.
14. The system shall provide a database for music selection.
15. The system shall provide administrators with song-list processing and upload functionality for music using a CSV file. These songs will be added to the database for future use.
16. The system shall provide administrators functionality to modify the competition schedule after successful creation for any student to resolve external conflicts/concerns.
17. The system shall provide administrators functionality to manually add teachers who were not automatically uploaded via a CSV file.
18. The system shall provide administrators functionality to manually add new administrators when actively logged in.
19. The system shall provide administrators functionality to input students' scores after they are received from judges.
20. The system shall efficiently uploading documentation such as music lists or CSV files.
21. The system shall schedule students based on their age, level, and preferences.
22. The system shall support concurrent competitions.
23. The system shall provide administrators with the ability to assign volunteer tasks to teachers.
24. The system shall use Google's Calendar API for an events calendar that will be managed solely by the chairman/chairwoman.
25. The system shall implement JSON web tokens (RFC 7519) to authorize and protect users.

2.2 Non-Functional Requirements

The following non-functional requirements are “constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process, and constraints imposed by standards. Non-functional requirements often apply to the system as a whole rather than individual system features or services” [10].

1. The system shall use Google’s Material Design standards to assist with UX/UI.
2. The system shall use only responsive web-pages that are viewable on multiple platforms.
3. The system shall utilize modern web technologies, front-end frameworks, back-end frameworks, and server-side frameworks to provide a platform feasible for future business requirements.
4. The system shall use Node [4] and Express [5] for server-side hooks and handling, and PostgreSQL for database management.
5. The system shall use modern security techniques (encryption, salt, hashing, role-based access control, etc.) to protect user confidentiality and privacy.
6. The system shall implement input-validation, form-validation, scheduling-validation, and other methods of validation to ensure that the event-scheduler, document-generator, and other services work properly at each step.
7. The system will be functional, and viewable for all modern browsers.
8. The system shall confirm registration by verifying payments with PayPal’s Braintree API.
9. The system shall use TLS 1.2 (RFC 5246) or TLS 1.3 (RFC 8446) and digital certificates for secure end-to-end network communications.
10. The system shall be highly available.

2.3 Detailed Use Cases

The use cases defined below in Figure 1 “identify the individual interactions between the system and its users other systems” [10], or “actors involved in an interaction and names the type of interaction” [10]. Those listed primarily concern interactions from systems re-implemented from previous versions of ARIA (1.0/2.0) into ARIA 3.0 such as interactions between customers and the customer portal shown in Figure 2, and interactions between administrators and administrative functions like event creation shown in Figure 5.

1. **CreateAnEvent**
   Administrators can create an event by providing initial information about the event such as the event name, event date, event start time, event end time, event location, and event fees. Registration forms and tables from the PostgreSQL database will be created to manage the event.
2. **ModifyAnEvent**
   Administrators can modify an existing event by retrieving it from the PostgreSQL database. All fields from a created event (event name, event date, event start time, event end time, event location, and event fees) will be mutable.
3. **GenerateEventDocuments**
   Administrators can generate event documents to advertise an active event and help customers, students, and teachers to find their respective destinations.

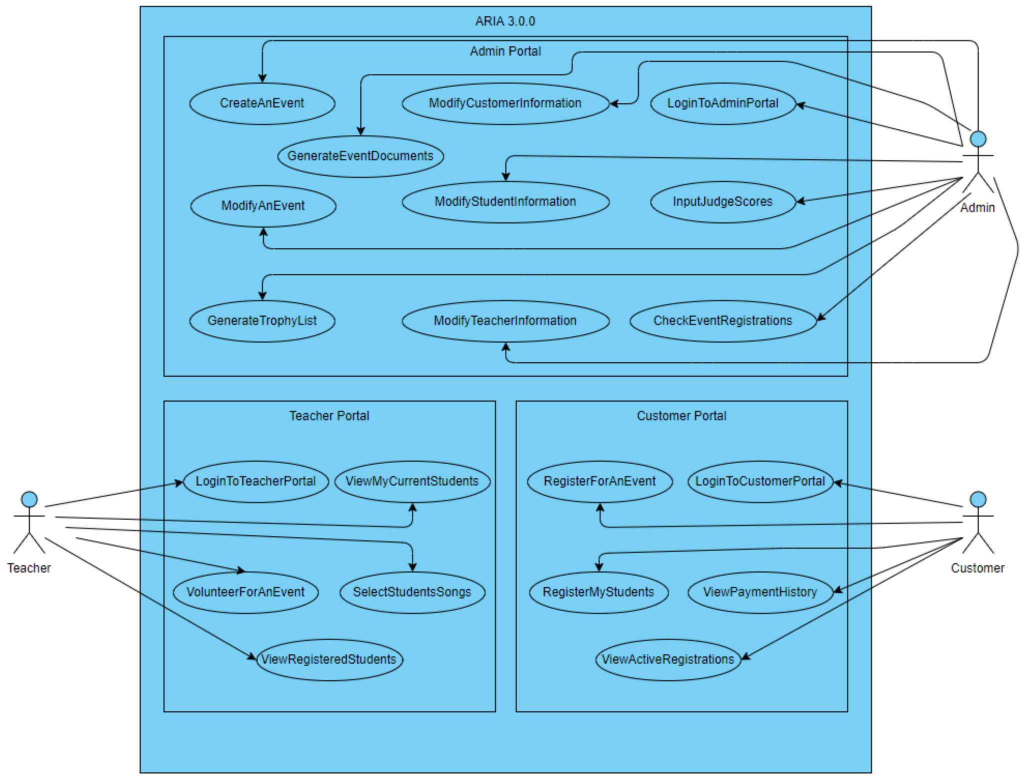


Figure 1: A use case diagram displaying the users of the system and the system functionalities related to each user along with their respective portals in the web-application service.

4. **GenerateTrophyList**
   Administrators can generate a trophy list for participants of an event. Students who receive a score of Superior, Superior with Distinction, or Command Performance will be added to the trophy list for an event.
5. **ModifyCustomerInformation**
   Administrators will be able to assists customers by modifying or deleting their information as necessary to maintain ARIA's list of active users in the PostgreSQL database.
6. **ModifyStudentInformation**
   Administrators will be able to assists students by modifying or deleting their information as necessary to maintain ARIA's list of active students in the PostgreSQL database.
7. **ModifyTeacherInformation**
   Administrators will be able to assists teachers by modifying or deleting their information as necessary to maintain ARIA's list of active teachers in the PostgreSQL database.
8. **LoginToAdminPortal**
   Administrators will be directed to a landing page with a login form where they will present

their credentials. Valid credentials will be authenticated allowing administrators access to the administrator-portal.

9. **InputJudgeScores**
Administrators will be able to input the scores of a student after their performance has concluded. Judges will provide a student's scores to an administrator after the student has performed. When ready, the administrator will be able to save a student's scores to the PostgreSQL database.
10. **CheckEventRegistrations**
Administrators will be able to check all student registrations for an event. Administrators will be presented with information indicating whether a payment is still "pending" or has been "confirmed" as well as whether a teacher has submitted a student's songs and finalized their registration.
11. **LoginToTeacherPortal**
Teachers will be directed to a landing page with a login form where they will present their credentials. Valid credentials will be authenticated allowing teachers access to the teacher-portal.
12. **ViewMyCurrentStudents**
Teachers will be able to view their current students. Students information including contact information, music-level, and the event they most recently participated in will be shown.
13. **ViewRegisteredStudents**
Teachers will be able to view students with a registration for an upcoming event. Information about the event with respect to registered students will be displayed to provide teachers with immediate access to event information.
14. **SelectStudentsSongs**
Teachers will be able to select the songs for students who have successfully paid their registration fees. After song-selection, teachers will be able to confirm the registration as complete, providing a notification to administrators.
15. **VolunteerForAnEvent**
Teachers will be able to volunteer for an event. Various duties are required for events such as door- monitor, and greeter. Teachers who elect to volunteer will be able to choose a duty related to the event. Once the volunteer process completes, administrators will be notified.
16. **LoginToCustomerPortal**
Customers will be directed to a landing page with a login form where they will present their credentials. Valid credentials will be authenticated allowing customers access to the customer- portal.
17. **RegisterMyStudents**
Customers will be able to access a page in their portal allowing them to register students. Information about each student will consist of the students first name, middle initial (optional), last name, music level, and music teacher. These criteria will assist the event-scheduler during the registration process.
18. **RegisterForAnEvent**
Customers may register any student on their account for an event. The event scheduler guide customers through the registration process. After completing the registration process the registration will be stored in the PostgreSQL database.

19. **ViewActiveRegistrations**
Customers whose payments have successfully been processed will be able to view any active registrations for upcoming events. Data for the event as well as the registered individual will be retrieved from the PostgreSQL database.

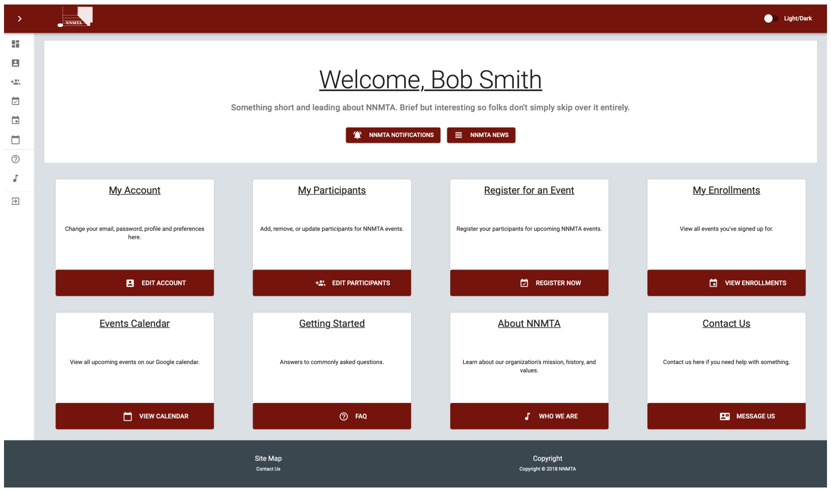
20. **ViewPaymentHistory**
Customers whose payments were processed successfully will be displayed to ensure that no errors were made during payment-processing. Viewable information will include the payment method, payment date, payment amount, the event being paid for, and the student participating in the event.

3 Results

One major feature of the web-application service that we implemented was a portal was for each primary user (customers, teachers, and administrators). For example, the customer portal can be seen in Figure 2. Another feature of the web-application service in ARIA 3.0 is payment processing. Payment processing was implemented using PayPal's Braintree API as shown in Figure 3. Finally, a third feature implemented in ARIA 3.0 is the document generator. The document generator was added as a separate service and one of the documents it generates, the announcing sheet, is shown in Figure 4.

ARIA 3.0 was built using a set of modern web frameworks, libraries, tools, and technologies. For our web application service, we primarily used the following technologies: React [7], Redux [1], Node [4], Express [5], and PostgreSQL. For our document generator, a separate service, we utilized Python, and Flask [9] to process and generate requested documents.

To support containerized application development our team utilized Docker [6] and Alpine Linux (a light-weight, security-focused distribution). We integrated our containerized application with Kubernetes [3], CircleCI [2], Git, and GitHub to automate, orchestrate, test, and manage our application. Finally, ARIA 3.0 implements a role-based access control security approach and session management using JSON web tokens (RFC 7519) and passport, a Node [4]



Welcome, Bob Smith

Something short and leading about NNMTA. Brief but interesting so folks don't simply skip over it entirely.

NNMTA NOTIFICATIONS
NNMTA NEWS

My Account
Change your email, password, profile and preferences here.
EDIT ACCOUNT

My Participants
Add, remove, or update participants for NNMTA events.
EDIT PARTICIPANTS

Register for an Event
Register your participants for upcoming NNMTA events.
REGISTER NOW

My Enrollments
View all events you've signed up for.
VIEW ENROLLMENTS

Events Calendar
View all upcoming events on our Google calendar.
VIEW CALENDAR

Getting Started
Answers to commonly asked questions.
FAQ

About NNMTA
Learn about our organization's mission, history, and values.
WHO WE ARE

Contact Us
Contact us here if you need help with something.
MESSAGE US

Site Map
Contact Us

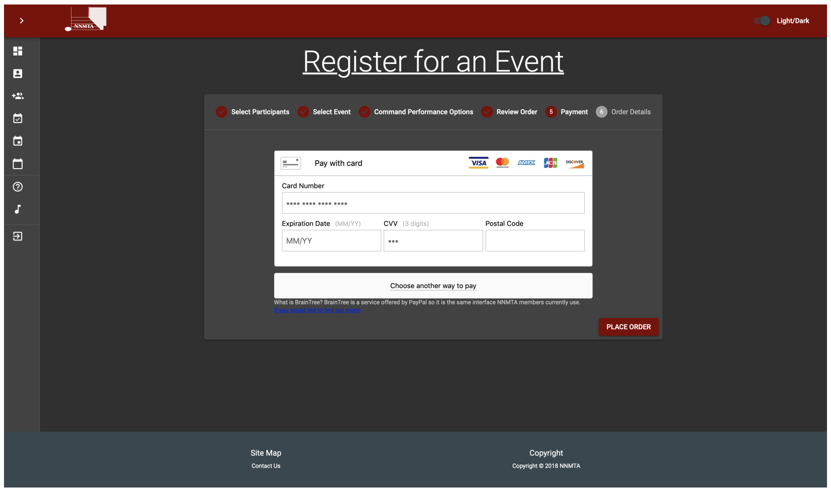
Copyright
Copyright © 2018 NNMTA

Figure 2: The customer portal dashboard (light-theme) which contains a set of cards. Each card describes a function of the customer portal to the user.

package compatible with Express [5].

Portals: Each primary type of user (customer, teacher, and administrator) has a portal which provides that user with information regarding their account and the set of functionalities that user is allowed to perform. For example, the customer portal displayed in Figure 2 contains functionalities for adding event participants (students), registering for events, viewing active registrations, etc.

Payment Processing: Payment processing for event registrations are performed using PayPal's Braintree API. The Braintree API was used because it supports major credit cards and PayPal, a drop-in UI (illustrated in Figure 3), a testing suite for verifying functionality, and the API conforms with modern financial regulations. Using PayPal should aid in the adoption of ARIA 3.0 by customers when paying for services provided by NNMTA.

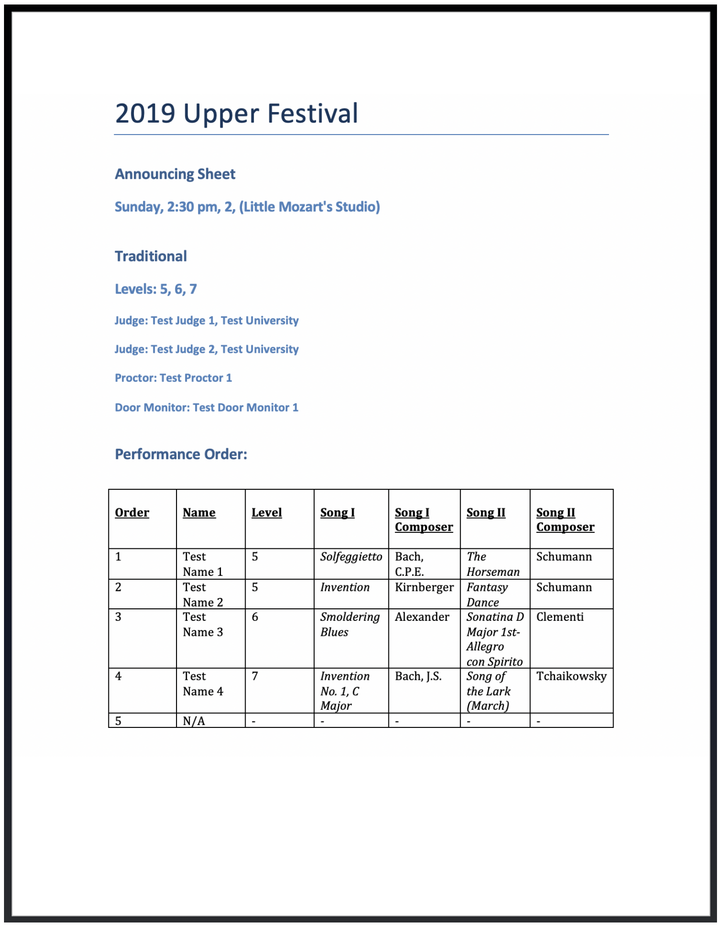


What is BrainTree? BrainTree is a service offered by PayPal so it is the same interface NNMTA members currently use.

Figure 3: PayPal's Braintree drop-in UI is shown (dark-theme). It is the fifth step of event registration and it is a feature of the customer portal.

Document Generation: Document generation was built as a separate service to ARIA 3.0 using a Python-Flask web server. The document generator listens for incoming processing requests and accepts a JSON object from ARIA 3.0's web-application service containing the document types to be generated and customer, teacher, and event-data to fill in the document fields. This data is gathered from ARIA 3.0's PostgreSQL database. One of the documents that ARIA 3.0 generates, announcing sheets, are shown in Figure 4.

Event Creation: Event creation is a primary function of administrators. A form, shown in Figure 5, is presented to administrators which helps to set up the event criteria. Once created, the event goes to the scheduling phase where it can either be scheduled and propagated to customers and teachers or the criteria can be modified.



2019 Upper Festival

Announcing Sheet

Sunday, 2:30 pm, 2, (Little Mozart's Studio)

Traditional

Levels: 5, 6, 7

Judge: Test Judge 1, Test University

Judge: Test Judge 2, Test University

Proctor: Test Proctor 1

Door Monitor: Test Door Monitor 1

Performance Order:

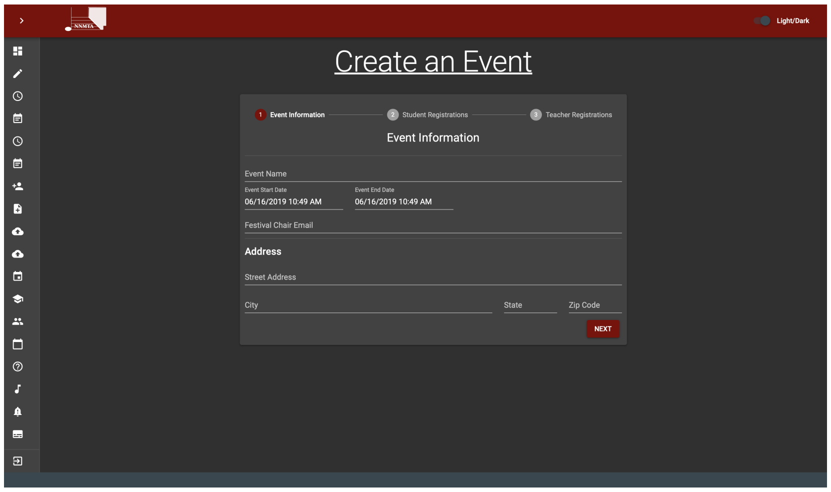
Order	Name	Level	Song I	Song I Composer	Song II	Song II Composer
1	Test Name 1	5	*Solfeggietto*	Bach, C.P.E.	*The Horseman*	Schumann
2	Test Name 2	5	*Invention*	Kirnberger	*Fantasy Dance*	Schumann
3	Test Name 3	6	*Smoldering Blues*	Alexander	*Sonatina D Major 1st-Allegro con Spirito*	Clementi
4	Test Name 4	7	*Invention No. 1, C Major*	Bach, J.S.	*Song of the Lark (March)*	Tchaikowsky
5	N/A	-	-	-	-	-

Figure 4: An announcing sheet created by the document generator with entries in the documents fields from information gathered from ARIA 3.0's database and passed to the document generator as a JSON object.

4 Conclusions and Future Work

ARIA 3.0 is a full-stack single-page web-application which demonstrates that software currently hosted on a platform such as WordPress can be developed as a proprietary stand-alone microservice application. Migrating features from previous versions of ARIA (1.0/2.0) to ARIA 3.0 was accomplished by decomposing the previous applications features into loosely-coupled services. Frequent meetings with stakeholders were an integral part of determining which business capabilities should be fulfilled by ARIA and implemented as services.

In total, our team was able to implement three portals, payment processing, document generation, and the infrastructure for future development. Despite our progression, further work needs to be done before ARIA 3.0 can fulfill its stated goal of replacing the current version (ARIA 2.0). Some of this work includes: the event scheduler (currently in development), and new features such as song-list processing, online judge scoring, short message service (SMS) notifications, and email notifications.



Create an Event

1 Event Information — 2 Student Registrations — 3 Teacher Registrations

Event Information

Event Name

Event Start Date: 06/16/2019 10:49 AM
Event End Date: 06/16/2019 10:49 AM

Festival Chair Email

Address

Street Address

City | State | Zip Code

Figure 5: The first step of the event creation form is shown (dark-theme). This form helps administrators create new events once logged into ARIA 3.0's administrator portal.

References

[1] Dan Abramov and the Redux documentation authors. Redux, A predictable state container for JavaScript apps. https://redux.js.org/, 2015. Last Accessed June 27, 2019.

[2] Circle Internet Services, Inc. CircleCI: The shortest distance from idea to execution. https://circleci.com/, 2019. Last Accessed June 27, 2019.

[3] Cloud Native Computing Foundation (CNCF). Kubernetes, Production-Grade Container Orchestration: Automated container deployment, scaling, and management. https://kubernetes.io/, 2019. Last Accessed June 27, 2019.

[4] Node.js Foundation. Node.js, a JavaScript runtime built on Chrome's V8 JavaScript engine. https://nodejs.org/en/, 2009. Last Accessed June 27, 2019.

[5] Node.js Foundation. Express: Fast, unopinionated, minimalist web framework for Node.js . http://expressjs.com/, 2017. Last Accessed June 27, 2019.

[6] Docker Inc. Docker: Enterprise Container Platform for High-Velocity Innovation. https://www.docker.com/, 2019. Last Accessed June 27, 2019.

[7] Facebook Inc. React: A JavaScript library for building user interfaces. https://reactjs.org/, 2019. Last Accessed June 27, 2019.

[8] David A. Mar Jr., Emily Huang, Vladislav B., Savranschi Renee T. Iinuma, Wesley Kepke, Ernest Landrito, Kyle Lee, Cynthia R. Harris, Sergiu M. Dascalu, and Frederick C. Harris Jr. ARIA: Efficient Music Festival Manager. In Frederick C Harris Jr., Sergiu M. Dascalu, and Sharad Sharma, editors, *Proceedings of the ISCA 26th International Conference on Software Engineering and Data Engineering (SEDE 2017)*. ISCA, 2017. https://www.cse.unr.edu/~fredh/papers/conf/181-aemfm/paper.pdf.

[9] Armin Ronacher. Flask: web development one drop at a time. http://flask.pocoo.org/, 2019. Last Accessed June 27, 2019.

[10] Ian Sommerville. *Software Engineering*. Pearson, 10th edition, 2015.