Group member details: names and SIDs

Our group will consist of three members. Bradley Thurlow (44779496), James Ridley (44805632) and Bryce Altman (44914792).

Project title

BookBarter

Introduction

BookBarter in its simplest form allows authors, publishers and bookstores to distribute its ebooks to various consumers on the network. BookBarter essentially allows for businesses to connect to it, sign up and start distributing a book to anyone who is using the service. BookBarter will also aid in facilitating the back end purchase of the e-books on its platform. This means that any purchase made via the site will all be handled via BookBarter so the vendor can focus on what they do best, writing and selling books.

Design

System Overview and Details

The core philosophy of microservices is to build independent services, decoupled in nature which all behave as their own service. These services must expose their service via an API which can be consumed by other services or the client. These services all handle their own business logic and perform their tasks as instructed and more importantly when required. Essentially these services all work independently and can run fine without any outside input. At one level of abstraction it will appear that all of our decoupled services work as one entity but in reality it is a number of services connected, consuming each other via their API's when needed. Each service will have its own primary function and due to its strength in modularity, they will perform only their desired task and nothing more.

Our services can be broken up into two distinct categories, the front end and the back end. These backend services will encompass the bulk of all the business logic. These services will all expose a ReST API in which other services or the client which can consume it. These services will communicate via HTTP requests as each request is generated. Since the services are decoupled it will be important that each service knows how to communicate with its other services, if so required.

BookBarter will have four distinct microservices in the backend. These services will all contain their own database. These services will communicate with the front end as they are required by the client. BookBarter will contain a service for the following: web server, browse, login and favourites. These services will encompass the backend portion to our application. The front end will contain the ui which acts as the interface for the backend services, this has been implemented using the angular framework. Our system will also encompass authorisation which has been implemented using JWT tokens. These components will discussed in depth under implementations.

These technologies have been incorporated into docker containers in which we can build the service. This is important in allowing us to deploy many instances of our applications if we need to but has also been important in allowing us to develop the application, as each docker instance contains the necessary components we need to run the work and constantly don't need to set up our development environment everytime on different machines.

Another key component to our system is the databases. Since microservices embraces a decoupled emphasis in terms of its architecture, the databases gain a significant boost to their performance, reliability and accuracy as each database schema can be designed distinctly for that services needs. This allows us to choose the best services suited for that service. This feature will also be important in reducing latency and increasing system health as it will minimise cross-domain calls.

The design philosophy we chose to follow as a team was one which should encompass a multitude of attributes, focusing on areas of friendliness, simplicity and the end users needs. We wanted to make this platform so intuitive that it felt like our customers have been using it for years and couldn't picture their lives without it in the future.

System Components:

BookBarter follows the microservices architecture, which means our backend will be divided into many smaller components which all work independently to create the application as a whole. The first step in the design of the application was to split up all the separate aspects to determine which features require or deserve their own separate microservice, and which ones could be integrated together.

The first microservice that was implemented was the web server, for obvious reasons. Without it, a user could not perform any actions such as registering, logging in, or even

browsing for books. The next highest priority to us as a team that needed its own microservice was the register/login functionality. This is most likely the first action to happen when any user visits the page, so it made sense for us to implement this next. The login microservice handles everything associated with a users profile, most notably registering a new user and storing their details in the database, and handling the authentication of existing users to ensure that they have proper access to our application.

The next microservice that was implemented was for the browsing functionality. This microservice handles things such as reading books from the database and displaying all the information, writing to the database if a book is added by an admin account, and even searching for specific books.

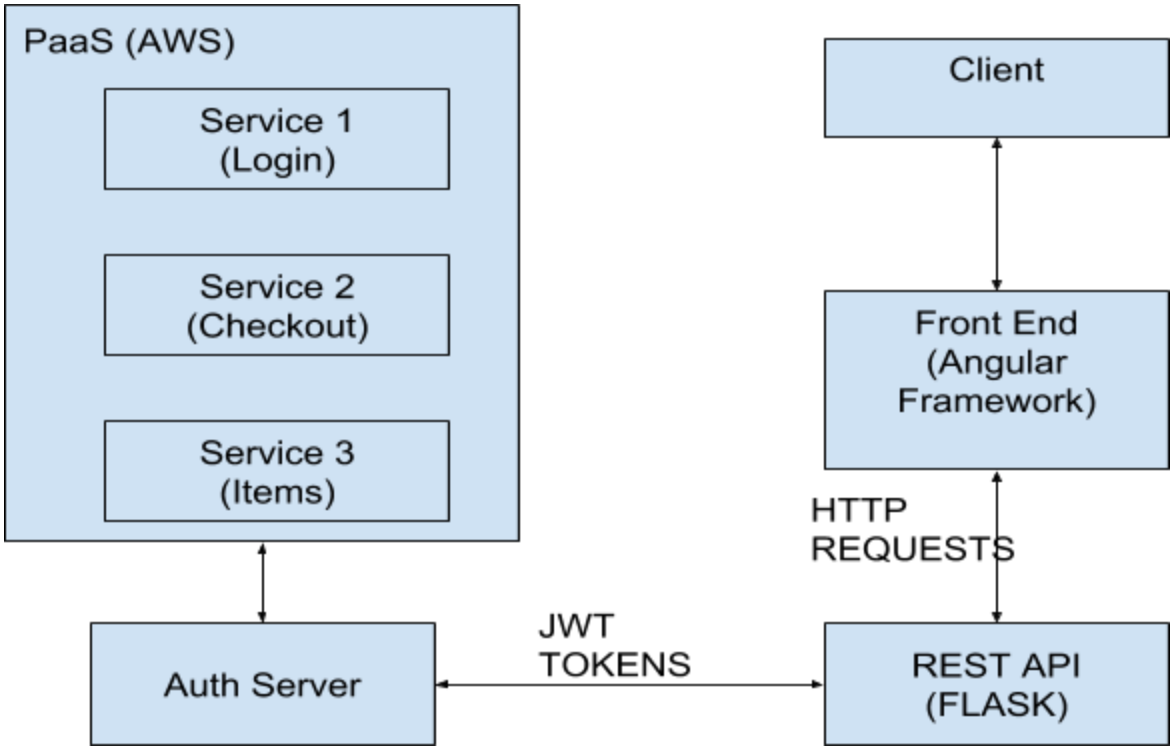
The final microservice for our application was originally intended to be a checkout system. However, as our application changed and evolved over time, we decided that a traditional checkout was not necessary, as users have access to a certain number of books over time based on their subscription. To substitute this checkout feature, we decided that a favoriting system would be best, so that users can save and access their favourite books in one simple location.

The front end of our application we decided to go with a component based architecture for the design, as we felt that this category of design complemented the microservices architecture the best. This means that we were able to split up each aspect of the user interface and have each of them as their own independent component, and therefore, easily connect them to an individual microservice if required. Selecting this architecture allowed our front end to have cleaner, faster and more robust development, which are all aspects that we were striving for in this project. Another advantages of this component based design was that it allowed for the reuse of components, which again saved time for our whole team which was a crucial resource over the life of this project.

In the planning phase of our project we decided that three different types of testing were crucial to the consistency of our application. The first being unit testing, to ensure that all our methods are correct and working as expected throughout the implementation. These tests will give our application a solid foundation on which to build upon. The next was integration testing, to confirm that multiple methods behave appropriately when integrated together, as sometimes when individual methods come together in a whole system, there can be some unexpected behaviour taking place. The final form of testing we decided to make use of was end to end testing. These are automated tests that essentially simulate a real user performing task on the actual application. They are very

useful for ensuring that our application behaves as we expect after a new feature is implemented, or even identifying logic errors we may have overlooked.

System architecture



Implementation

Front End:

The front end of the applications has been built very simply. We tried to incorporate many aspects of user interface design and encompass many practises in which we believe would lead to a good user experience. We intended at maximising usability and the user experience within the app, with the aim of making the users interactions as simple and efficient as possible.

The front end of our application is implemented and built using the angunal framework. The angular framework has allowed us to implement a combination of html, css and javascript in a timely manner. Angular has been crucial to our development process as it has allowed us to the maximise speed in development. It has allowed us to build features quickly with its incredible tooling capabilities, with the integration of simple, declarative templates. Angular also has a two very important components which are

paramount for a microservice type application and that is firstly the ability to develop and build applications once and then deploy it to any target be it web, mobile, native mobile and native desktop. Secondly allows us to achieve maximum speeds possible on our web platform as we can easily incorporate web workers and server side rendering. This framework was also chosen for future proofing the application as it can be easy to scale and duplicate.The angular framework also works well when it comes to testing. We have integrated karma unit tests to check if components of our platform is behaving in the manner that it should be.

Node.js will make our apps extremely fast and efficient. Angular will be served with Node.js and this process will happen quickly due to the features such as auto reloading and hot swapping. Node.js is extremely efficient for serving REST APIs and allows for asynchronous communication incorporating a style in which our architecture aims for, whilst incorporating an event-based architecture appropriate for BookBarter.

Back-End:

Flask framework and python will help us develop the appropriate services and make them consumable by the client and other services via their RESTful APIs. The services will all be programmed using flask as it offers many powerful tools without the need for a fully fledged implementation like Django. Flask is extremely beneficial when it comes to aspect such as routing as it will help in processing the various HTTP request sent by the client. Flask is a light framework however it is very powerful in its nature and because of this it is also very popular. Due to its popularity we were able to create a faster development environment, easier to read, write and maintain code. Since we are only a small team we thought these benefits were paramount in getting a project deliverable in such a short timescale as its learning curve was not as vast as other frameworks. Flask and python alike also have many packages and libraries to choose from in case we chose to add in extra features or need other components from other libraries they are easily accessible, and due to its popularity you know that the libraries will not undergo any major changes unlike other popular languages. Flask provies many easy to use extensions which allowing for easy database integration, form validation, upload handling and of course authentication. Flasks simplicity is one of its biggest strengths it may be labelled as a ‘micro’ framework but in essence it is only a micro framework because it leaves all the decision up to the developer. It keeps its core simple but extensible, this means we as a team are able ultimately choose the packages, databases and architecture on our terms.

The database component of any microservice is one of its most paramount features. Each of the services contain their own database. Since they contain their own database it allows us to create a separate schema for each service thus allowing us to create different attributes and fields for the logic they need. This is an extremely efficient way of creating a database as it will ensure data correctness, decrease error states and load reduction. For our database we choose to use MariaDb. We choose to use MariaDb as it is an open source and powerful RDBMS. We thought this would be very handy as it essentially behaves and works just like MySQL however we could add many external packages to it if we needed them. Since these packages are free we would not be able to get the same level of support for MySQL unless we got the enterprise edition. Since MariaDb is growing in popularity we knew that only more and more add ons would become available in the future, especially in regards to other large corporations adopting it. Since we wanted this application to scale if it needed we would need access to many of the enterprise MySQL offers. Services such as threadpooling are offered via other open source collaborators for the MariaDb framework. MariaDB eas full features and also contains many helpful tools in which we could test and monitor the databases as we needed.

Other:

Docker is a containerisation application which has been very useful in building this application. The containers also provide a layer of isolation over processes. Allows for the virtualization of ports which docker takes care of. By default flask ports to 5000 but with docker we can run many instances on virtualized ports which we don't need to worry about. Containers also allow all the different services to contain their own depencies, files and any other working business logic which they need. This allows each service to be decoupled from each other making them easily deployable depending on which instance of the application is in greater demand at that point in times. Containers have also been very important in our development pipeline. Containers have allowed us to build an instance of the application on our local machine to test if firstly everything runs as expected but also because it has allowed us to run each service without difficulties if we haven't got certain dependencies installed. This is very important especially when working in small teams such as this because each developer will have their machine configured in a different way and everyone will have different dependencies installed. This means that we could develop without worrying how each person has their machine configured. Docker compose has also been extremely important in testing our application whilst developing. Normally we would have to test our application as a whole thus deploying to a staging server and waiting

for everything to build to test out one feature but with compose we could speed up the process by deploying services locally and independently.

JWT Tokens will offer the authentication on our service. The tokens will allow users to authenticate themselves in order to gain access to restricted areas, which require these login credentials. Each of the microservices will also contain a JWT key which will be used as authentication for communication between services.

Discussion

The assignment had many challenges and obstacles in which we had to overcome in order to build the application. Firstly there was a tremendous learning curve which had to be overcome. Many aspects towards building services seemed foreign, with many aspects going in over our heads. After considerable nights researching we slowly understood certain aspects which could allow for progress towards our application. We quickly learned however that the more we knew the less we knew at the same time. There was so much information on all aspects of microservices, from the architecture, how to effectively run them, which stacks performed better etc… There was a plethora of information which was daunting and trying to choose the best options for our applications was a difficult task.

An initial challenge was setting up everyone's development environment. This was a major issue in the beginning since everyone was essentially working on different aspects and nothing would seem to work. As the services grew the more depencies that would be needed and different dependencies depending on the different services. This was an issue until we got docker working correctly. With docker and docker compose running we could all work on different aspects independently and build them with ease.

Time was another issue which deeply impacted the performance of our group as i'm sure did with many others. Time was against us and it was hard to maintain a steady rate of productivity in regards to deadlines and by important milestones in which we wanted to hit. These milestones slowly got pushed back and delivering certain aspects got more and more difficult especially when certain components wouldn't work when we were implementing their desired functionality. This time pressure in general, hard to manage the work in the time frame and with three full time students. Could've implemented more if we had more time, but with the benefit of the agile methodology we used we were able to stop development at any time and submit a working product. Ran out of time for implementing a number of different testing mechanisms in which we

aimed for, in particular we would've liked to spend more time on unit testing and integration testing. We have some basic tests working, but a more vigorous and thorough one would be implemented given more time.

We found the emphasis on documentation challenging, as a lot of our time was spent planning, designing, or documenting the application rather than implementing it. Usually in agile methodology there is less emphasis on planning so we felt a little limited in that regard.

Evaluation

The main way we experimented and analysed results within our project was the use if different kinds of testing. However, as mentioned previously, we did run out of time to include all of the tests we would have liked. From the tests that we did include everything seemed to be in working order. Our most successful category of testing was the automated end to end tests, by using selenium and protractor. In Angular protractor is what wraps around selenium in order to automate the web browser. Karma and Jasmine were the testing frameworks we used for the front end unit testing, and also were used to validate the end to end tests. For the backend, we simply used the python 3 unittest framework to write and run our back end unit tests. However, we would have liked to spend more time in this area.

Throughout the project we tested if various routing and HTTP requests were handled correctly via the postman api. We also tested these by running and staging builds in which we regularly tested if everything worked in the manner in which it was suppose to.

Conclusion

In conclusion we have gained a great deal of knowledge working on this assignment. Many of the aspects have not only benefited us in terms of the unit but a more general sense making us more industry ready. Projects like this are vital in working in industry no matter if the team is one person or large teams. Working with the various frameworks, the vast research and learning about these technologies have given us many vital insights into how projects are set up from start to finish. The technologies have been a fantastic in learning. Building the proposed project was not only fun but a great way to pick up on how things work in a more abstract view. The presentations and pitching of proposals has also worked well in replicating a workforce environment. Overall the experience has been very challenging but at the same time a very rewarding one.

Bitbucket project repository/wiki

Our repository for the assignment will contain all source code and documentation for BookBarter. Our repository can be found at: https://github.com/bradt6/BookBarter

References

Aggarwal, S. (2014). Flask framework cookbook.

Balalaie, A., Heydarnoori, A. and Jamshidi, P. (2016). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. IEEE Software, 33(3), pp.42-52.

Fowler, S. (n.d.). Production-ready microservices.

Goasguen, S. (2015). Docker Cookbook. O'Reilly Media.

NEWMAN, S. (2018). BUILDING MICROSERVICES. [S.l.]: O'REILLY MEDIA, INC, USA.

Venugopal, M. (2017). Containerized Microservices architecture. International Journal of Engineering and Computer Science, 6(11).