Software Documentation and Architectural Analysis of Full Stack Development

Clark Jason Ngo
clarkngo@cityuniversity.edu
City University of Seattle

Dr. Jin Chang
kyongchang@cityu.edu
City University of Seattle

Dr. Sam Chung
chungsam@cityu.edu
City University of Seattle

Abstract

The purpose of this paper is to demonstrate how to decrease the barrier to entry for open source projects and full stack web development. Documentation and architectural modeling are implemented to lessen the steep learning curve for open source and full stack web development. The research will bring in ten hands-on practices (HOPs) that would teach students to build a full stack web application. The research will conduct the following: 1) Create documentation and architectural model for the hands-on practice. 2) Evaluate the documentation and architecture model with a survey.

Keywords: Software Documentation, Software Architecture, Open Source, Full Stack, MEAN

1. PROBLEM AND MOTIVATION

Open source software invites anyone with the technical capability to contribute to the code base. However, most open source software has a big barrier to entry as there is no support for visual architectural modeling (Kim, Chung, & Endicott-Popovsky, 2014). Open source software is very complex. It has a steep learning curve to be able to contribute or use it. The non-existence of documentation leads to further complexity. Full stack web development is difficult to learn as well (Shah, & Soomro, 2017).

A study by (Aghajani, et al., 2019) showed issues in software documentation. A sample size of 955 document issues were surveyed in the study that includes StackOverflow, GitHub, and mailing lists. 88% (840 issues) were GitHub issues and pull requests. 50% (426 issues) out of the 88% (840 issues) were information content: "what". 53% (227 issues) were about missing or poor document, and missing diagrams. Breakdown of document issues: 50% on What (includes documentation and diagram) 27% on Information Content (How), 8% on Process Related, and 14% on Tool Related.

A study by (Shah, 2017) showed difficulties in learning Node.js, server-side component of MEAN stack. A sample size of 80 developers were part of study to answer survey questions. The following shows the questions and the results: (1) Learning of JavaScript for Node.js was a challenge: 23.9% felt learning challenging and 44.8% felt learning a little bit of challenge. (2) Learning of JavaScript for NoSQL Databases was a challenge: 31.3% felt learning challenging and 20.9% felt learning a little bit of challenge. (3) Event-Driven of Node.js challenging: 34.3% felt

learning challenging and 25.4% felt learning a little bit of challenge. (4) Non-Blocking I/O feature of Node.js challenging: 31.3% felt learning challenging and 26.9% felt learning a little bit of challenge. (5) Asynchronous Processing feature of Node.js challenging: 38.8% felt learning challenging and 17.9% felt learning a little bit of challenge.

How can we reduce the steep learning curve from both Open Source and Full Stack Development with MEAN Stack?

2. BACKGROUND

Architectural Model

Unified Modeling Language (UML) notation is a language that uses pseudo-code, actual code, pictures, diagrams and others to help describe systems. UML has advantages such as: (1) It will not be misunderstood because it's a formal language, (2) straightforward and concise, (3) comprehensive to describe all components of a system, (4) scalable because it is usable for small to big systems, (5) standard for vendors and academics. However, as UML is an abstraction to provide high-level overview, the notation will not describe the small details. Still, UML is better than detail overloading from modeling with code and ambiguity from modeling with informal language (Miles & Hamilton, 2006).

UML Documentation improves the software maintenance. With UML providing overview and structure, code changes can be quickly located, and solving tasks is easier. On the contrary, building UML documentation costs extra time to build and not useful for easy and small tasks to fix (Arisholm, Briand, Hove & Labiche, 2006).

Full Stack Web Development - MEAN Stack

MEAN stack consists of four technologies used together. These are MongoDB (Data Tier), Express (back-end framework), Angular (front-end framework), and Node.js (back-end environment). MEAN stack is a technology that uses JavaScript across all components removing the need for translation and saves build time (Dunka, Emmanuel, & Oyerinde, 2018). See Figure 1 for diagram on client-side, server-side, and database using JavaScript.

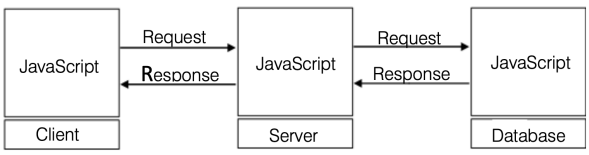


Figure 1 End-to-end JavaScript (Shah, 2017)

Back-end with Node.js

Node.js is built for handling asynchronous I/O while JavaScript is has a event loop built-in for the client-side, see Figure 2 for Node.js processing model.

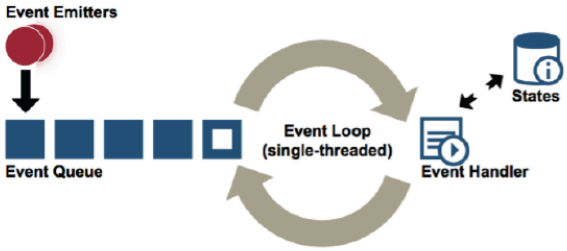


Figure 2 Node.js Processing Model (Esostalgic, 2014)

These are the components that makes Node.js fast in performance compared to other environments. However, the event-driven/callback approach makes Node.js difficult to debug and learn as well. See Figure 3 for Node.js System.

Node.js includes modules such as mongoose, which is a MongoDB object modeling, and express web application framework. Through node modules, abstraction can be achieved, which reduces the overall complexity of the MEAN stack.

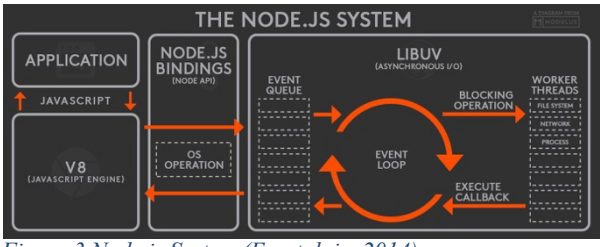


Figure 3 Node.js System (Esostalgic, 2014)

Back-end with Express Framework

Express is a minimalist and unopinionated application framework for Node.js. It is a layer on top of Node.js that is feature-rich for web and mobile development without hiding any Node.js functionalities (Adhikari, 2016).

Front-end with Angular

Angular is a web development platform built in TypeScript that provides developers with robust tools for creating the client side of web applications. It allows development of single-page web applications where content changes dynamically based on user behavior and preferences. It features dependency injections to ensure whenever a component is changed, other components related to it will be changed automatically. MVC (Model View Controller)

architecture, see Figure 4, can be applied with Angular (Adhikari, 2016).

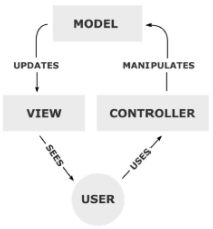


Figure 4 Model View Controller (Shah, 2017)

Database with MongoDB

MongoDB is a NoSQL database which stores data in BJSON (Binary JavaScript Object Notation). MongoDB became the de facto standard database for Node.js applications to fulfill the JavaScript everywhere using JSON (JavaScript Object Notation) to transmit data across different tiers (front-end, back-end, and database) (Adhikari, 2016).

Learning Curve

When we learn, we acquire new knowledge and skill. The rate of knowledge or skill acquisition is called the learning curve (Kang, & Hahn, 2009). Learning is defined as obtaining knowledge through studying and experiencing a subject matter. The three concepts of learning according to (Britto, R., Šmite, D., & Damm, L.-O., 2016) are knowledge, skill, and competence. Knowledge is information processed and absorbed through structured or unstructured learning. Skills are acquired through practice. Competence is the effectiveness of an individual to a certain field. Reducing a steep learning curve increases the return-on-investment (ROI) for software development as it will lead to faster building of new features and fixing or bugs. It also increases the productivity for developers who mentor (Tüzün & Tekinerdogan, 2015).

Learning curve becomes steep when the topic to be learned is complex such as software architecture that has challenges on understanding the theory behind it (design principles, tradeoffs, architectural patterns, etc.), visibility at scale only, and requiring communication between different stakeholders. The following principles were introduced to address the challenges: (1) embrace open source, (2) embrace collaboration, (3) embrace open learning, (4) interact with architects, and (5) combine breadth and depth. The approach for tackling challenges were (1) apply theory to practice, (2) contribute to the system, (3) integrate architectural views, and (4) providing feedback to other students. Additionally, reducing the steep learning curve can be done through mentoring. The use of markdown was implemented to facilitate sharing, versioning, and reviewing various systems (Van Deursen, et al., 2017). Another study found another way to reduce the steep learning curve through pairing developers with mentors. The study showed that activities in an Open Source Project (OS) with mentored developers were significantly higher than non-mentored developers (Fagerholm & Sanchez, 2014).

Documentation

Documentation helps define what a project do, why the project is useful, how users can start using the project, where users can get help, and also defines who maintains and contribute to the project (Prana, Treude, Thung, Attapattu, & Lo, 2018). Why is documentation important? There are dangers of internet search instead of looking at software documentation. Users might use the wrong information. Creators given an unsatisfactory for software built due to lack of good documentation (Van Loggem, 2014).

Documentation on Open Source

In open source software, documentation is mostly created using a markdown file called README.md. README is the custom standard platform used for software distribution. This markdown file is analogous to a website's home page. If the README.md is poorly written and maintained, developers would not be attracted to try out the technology. See Figure 5 for comparison of README.md file of Node.js in GitHub (Node.js, 2020) and Amazon Web Services (AWS) home page (Amazon Web Services, 2020).

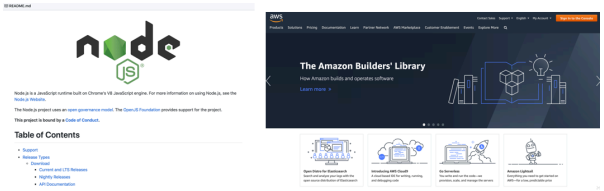


Figure 5 Open Source README.md versus Website Homepage

Amateur developers write documentation last. As most software has no set end date that it will be fully built and functional, you might end up forgetting to write documentation or have written it late the documentation is not as useful anymore. Professional developers write documentation first using Test-Driven Development or Behavior-Driven Development (Mastropasqua, 2016). Early and timely

documentation attract new developers to collaborate on the open-source projects and eases the on-boarding process to join it.

According to (Lee, 2018), best practices for creating and maintaining a software documentation are the following: (1) Include a README or text file as it is more human-readable then Hyper Text Markup Language (HTML). (2) Use a style guide to tell users how to use it. (3) Include a help command for developers without a Graphical User Interface (GUI). (4) Provide link to the full documentation to avoid cluttering the README file. (5) Apply version control to support versions of the project.

3. RELATED WORK

The key papers are compared with the availability of the topics: (1) documentation, (2) architecture, (3) open source, and (4) full stack. See Table 1 for summary of key papers.

Topic	[KRUC 95]	[KIM 14]	[ADMI 17]	[STAF 15]
Documen tation	No	Yes	No	No
Architecture [4+1 View]	Yes	Yes	Yes	Yes
Open Source	Yes	Yes	Yes	No
Full Stack [MEAN]	No	No	No	No

Table 1 Key Topics for Previous Work

The 4 + 1 View Model of Software Architecture

This paper by Philippe Krutchen built a UML architectural model using the 4 + 1 View Model, which consists of the Logical View, Development View, Process View, Physical View, and Scenarios. The Logical View supports functional requirements and used when the design method is object-oriented. The Development View shows the static organization of software in the development environment. The Process View shows the flow of synchronization in the design. The Physical View shows the connection of software to hardware. Scenarios puts together the architectural models and validates them through the perspective of the end-user (Krutchen, 1995). See Figure 6 for 4 + 1 View Model.

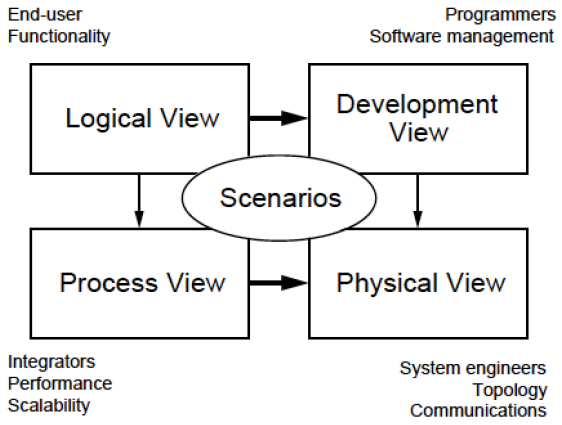


Figure 6 The 4 + 1 View Model

Software Architecture Model Driven Reverse Engineering Approach to Open Source Software Development

This paper by Kim designed architectural models for deployment view for MITREid Connect, an open source authentication protocol. See Figure 7 for Deployment View (Kim 2014).

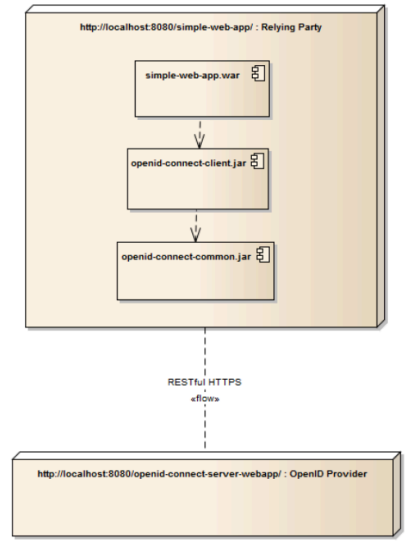


Figure 7 Deployment View (Kim, 2014)

He also developed a design view to show an overview to classes, object instances, and message exchanges. See Figure 8 for Design View (Kim, 2014).

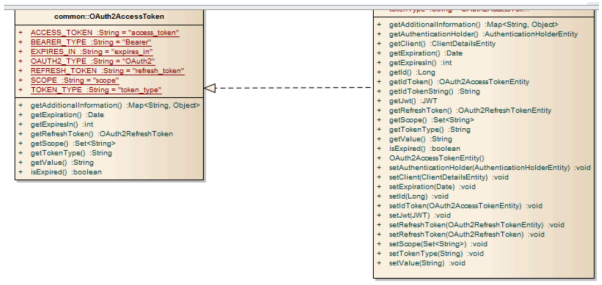


Figure 8 Design View (Kim, 2014)

Technical Design for Angular Apps

This paper by Hans Admiraal designed architectural diagram for Angular applications with UML notation and color coding for the package diagram to provide an overview to Angular modules and their dependencies to each other. See Figure 9 for package diagram (Admiraal, 2017).

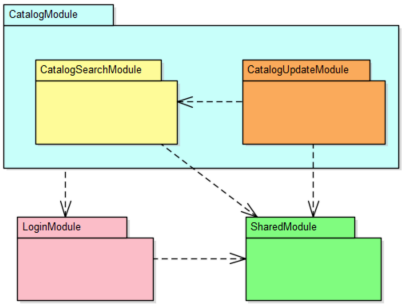


Figure 9 Package Diagram of Angular Modules (Admiraal, 2017)

He also designed a scenario-based sequence diagram to show the activity flow and color-coded based on the Angular modules. See Figure 10 for Use Case View example.

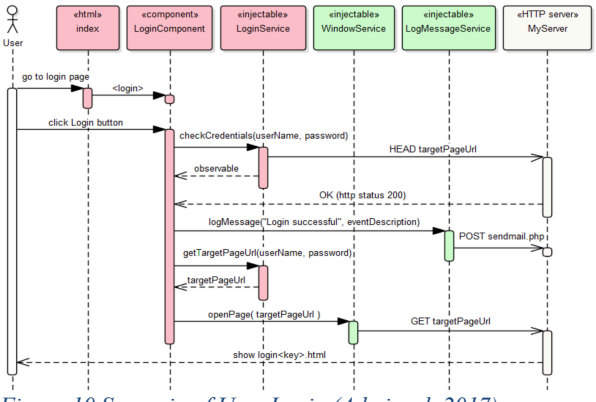


Figure 10 Scenario of User Login (Admiraal, 2017)

An article showed a request and request data flow sequence diagram. The diagram showed both the request and response across the MEAN Stack that includes the front-end, back-end, and database. See Appendix A for Request/Response Data Flow diagram (Stafford, 2015).

4. APPROACH AND UNIQUENESS

Topic	[KRUC 95]	[KIM 14]	[ADMI 17]	[STAF 15]	[NGO 20]
Documen tation	No	Yes	No	No	Yes
Architecture [4+1 View]	Yes	Yes	Yes	Yes	Yes
Open Source	Yes	Yes	Yes	No	No
Full Stack [MEAN]	No	No	No	No	No

Table 2 Author's Approach

The author's approach was the combined implementation of Kruchten's 4+1 View, Kim's Design View and experiment validation, and Admiraal's color-coded diagrams. See Table 2 for the author's approach.

4.1 Experimental Design and Methods

The author used the online diagram tool, Lucidchart, to build the diagrams from scratch. These designs are validated with a software engineer who has experience building full stack applications.

Physical View with Deployment Diagram

The deployment diagram shows 3 servers: front-end, back-end, and database. In the front-end, we require the browser as angular applications are browser-based web applications. The back-end server hosts our Node.js with Express on top of Node.js. In Express, we have the application and mongoose on top of it. Express will handle the communication on both front-end and database. The database server only includes a MongoDB. JSON is utilized to communicate across servers. See Appendix B for deployment view of MEAN stack.

Deploying in production has many options to choose from. In the author's use case, we deployed the front-end in Amazon S3, the back-end in EC2 instance, and the database in MongoDB Atlas. See Appendix C for Production Deployment.

Process View with Sequence Diagram

When an http request is made, the front-end will be triggered an Angular will pick up the request. The request will be passed internally in Angular with Route sending a request for the view to View/Template. View/Template will request the Controller. The Controller will then create a http request to a RESTful (Representational state

transfer) endpoint to the Server Side, which is Express/Node.js. The request will then be passed internally with Express/Node.js from its Route to the Controller/Model. The Controller/Model will make a request through the Mongoose ODM to interact with the Database Server that has MongoDB. MongoDB will process the request and respond the callback to Express/Node.js. Express/Node.js sends a JSON response to the Angular Controller. Angular Controller would respond with a view. See Appendix D for MEAN stack request/response data flow

Development View with Package Diagram
In a book store application example, the package diagram shows the dependency of each Angular module to other Angular modules. See Appendix E, for Angular Module Package Diagram.

Logical View with Class Diagram
On the server side, the book store application has Book class the has the following attributes: title, isbn, author, picture, and price. It also has the following methods: addBook, fetchBooks, fetchBook, updateBook, and deleteBook. See Figure 11 for class diagram for Book class.

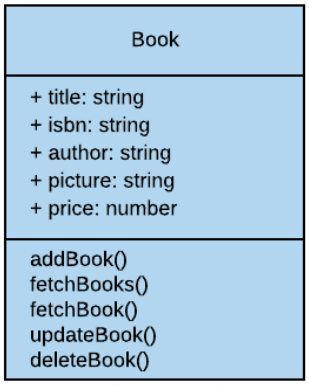


Figure 11 Class Diagram for Book Class

As our database is using a NoSQL database, the diagram will be shown with the JSON format. See Figure 12 for NoSQL diagram.

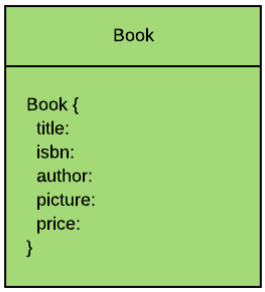


Figure 12 Diagram for NoSQL

Scenarios with Sequence Diagram

The scenario described is a user accessing a book store application. When the user enters the URL, JavaScript will be run and will hit the router of the front-end server, which is AppRoutingModule. AppRoutingModule will call the BooksComponent, which will load fetchBooks as its dependency injection. fetchBooks will then create an HTTP request to the back-end server that has a router, controller, and model to process the request and request to the database server. Database server processes the request and with the back-end server waiting, will grab the data and sent it back to the front-end server as a JSON response. The front-end will now have the data and the template view to show to the user. See Appendix F for scenarios of a book store application.

5. EVALUATION

The method to evaluate the impact of the documentation and architectural diagram in decrease the learning curve of full stack development is through a survey. The author used Google Forms to create two similar survey forms. They both ask for information about the participants, assessment before reading the documentation, and assessment after reading the documentation. The only difference with the two survey form is the documentation they will assess. One is assessing the official documentation for MEAN stack and the other is the author's documentation for MEAN stack. See Appendix G for author's documentation, see Appendix H for survey for official documentation, and see Appendix I for survey for author's documentation.

The evaluation method was through student research group participation, which might have potential bias due to the testers and test subject acquainted with one another. The survey had a small sample size that might not be considered a good representation.

6. RESULTS AND FINDINGS

For the survey on the official documentation, we surveyed 9 students who are in the computer science program with 5 students have 0-1 years of experience building software, doesn't know UML, Full Stack Web Development, and MEAN Stack. After reading the official documentation, the 6 students saying it increased there understanding of MEAN stack and the description and diagrams in the documentation was helpful. The official documentation got an overall rating of 4 out 5 for 8 students. The routing diagram was the most helpful in increasing their understanding. Most respondents suggested to

show more diagrams for explaining the technology. See Appendix J for the survey results of the official documentation.

For the survey on the author's documentation, we surveyed 6 students who are or had a degree in computer science. 3 students have 0-1 years of experience building software and 4 students have 2 or more years of experience building software. On average, 3 students have used and has work experience with UML, Full Stack Web Development, and MEAN Stack. After reading the author's documentation, the 5 students said that it significantly increased the understanding of MEAN stack and the description and diagrams in the documentation was very helpful. The author's documentation got 57% response of 4 out of 5 and 43% response of 5 out 5 for their overall rating of the author's. The restaurant analogy and process view sequence diagram were the most helpful in increasing their understanding. Most respondents suggested to add more description and more examples for the documentation. See Appendix K for the survey results of the author's documentation.

To compare the two results, the average return of high-level understanding to 15-minute time spent was used. It is calculated by using each result's average score of all responses divided by the max score of 5. No need to divide by 15-minute as the same amount of time was used for the documentation comprehension portion of the survey. The average return of high-level understanding to 15-minute time spent were 67% for the official documentation and 80% for the author's documentation. With this small sample size, it shows that author's documentation is 13% better than the official documentation for return on high-level understanding.

7. FUTURE WORK

With UML Notation, we can easily decouple the technology stack and add integrate new technologies. For author's future work, the MEAN Stack scenario view using sequence diagram was extended to add offline capabilities, which is one of the features of a Progressive Web App (PWA). PWA is an enhancement strategy to create cross-platform web application. See Appendix J for the Progressive Web Application Scenario View using Sequence Diagram. A physical view using deployment diagram was created for PWA as well. It shows how adding IndexedDB wrapper in the Angular application to communicate with IndexedDB, a client-side storage. It also shows service worker and cache storage to store data and be retrieved when the application's internet connectivity is offline. See Appendix L for the Progressive Web Application Scenario View using Sequence Diagram.

8. CONCLUSION

The learning curve from open source full stack web development is decreased with the help of high-level overview diagrams. The author's documentation is 13% more effective than the official documentation to decrease the learning curve. Documentation with architectural modeling adds additional time and resources to build. To decrease the time and resources to build these diagrams, high-level overview diagrams should be built instead of low-level. Most open source full stack development only have text descriptions and code base in their documentations, which make it too verbose. Implementing architectural modeling in the documentation of an open source full stack development gives a visualization that the developer can easily follow and digest the high-level concept. Thus, decreasing the learning curve for the developers.

8. REFERENCES

Adhikari, A. (2016). Full Stack JavaScript: Web Application Development with MEAN. Retrieved from: https://pdfs.semanticscholar.org/547a/f8ea205a8cc7c02506f58c9599a447da07a7.pdf

Adzic, G., & Chatley, R. (2017). Serverless Computing: Economic and Architectural Impact. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 884-889

Admiraal., H. (2017) Technical Design in UML for Angular Applications

Aghajani, E., Nagy, C., Vega-Márquez, L., Linares-Vásquez, M., Moreno, L., Bavota, G., & Lanza, M. (2019). Software documentation issues unveiled. *In Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press, 1199–1210. Retrieved from https://dl-acm-org.proxy.cityu.edu/doi/abs/10.1109/ICSE.2019.00122

Amazon Web Services. (2020). Amazon Web Services. Retrieved from: https://aws.amazon.com/

Arisholm, E., Briand, L., Hove, S., & Labiche, Y. (2006). The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation. *IEEE Transactions on Software Engineering*. 32(6), 365-381. Retrieved from: https://ieeexplore.ieee.org/document/1650213

Britto, R., Šmite, D., & Damm, L.-O. (2016). Group Learning and Performance in a Large-scale Software Project: Results and Lessons Learned. Retrieved from https://www.diva-portal.org/smash/get/diva2:1143810/FULLTEXT01.pdf.

Dunka, B., Emmanuel, E., & Oyerinde Y. (2018). Simplifying Web Application Development Using-Mean Stack Technologies. *International Journal of Latest Research in Engineering and Technology (IJLRET)*.

Esostalgic. (31, July, 2014). After much research and photoshop, I am proud to give you the finalized #nodejs System diagram. Retrieved from https://mobile.twitter.com/Esostalgic/status/494959181871316992

Fagerholm, F., Sanchez Guinea, A., Borenstein, J., & Munch, J. (2014). Onboarding in Open Source Projects. IEEE Software, 31(6), 54–61. Retrieved from: https://doi.org/10.1109/MS.2014.107

Fernandes, J., Gomes, R., Prada, R. & Ferrão, L. (n. d.) Softening the Learning Curve of Software Development Tools. Retrieved from http://www.joaofn.com/pdf/softening-the-learning-curve.pdf

Irshad, W. (2017). Five Common Mistakes in Agile. Retrieved from https://www.knowledgehut.com/blog/agile/five-common-mistakes-in-agile.

Kang, K. & Hahn, J. (2009). Learning and Forgetting Curves in Software Development: Does Type of Knowledge Matter? Retrieved from: https://pdfs.semanticscholar.org/05d1/a1ff8b17168bae748dca886ee4b3299f0dea.pdf

Kim, W., Chung, S., & Endicott-Popovsky, B. (2014). Software Architecture Model Driven Reverse Engineering Approach to Open Source Software Development. Proceedings of the 3rd annual conference on Research in information technology. Retrieved from https://www.tacoma.uw.edu/sites/default/files/sections/InstituteTechnology/W_Kim.pdf

Kruchten, P.B. (1995). The 4+1 View Model of architecture, IEEE Software. 12(6). 42 –50. Retrieved from: https://pdfs.semanticscholar.org/c5f4/9f3fe7624bd8ecfc5321d8e7b64a505b8f67.pdf

Lee, B. (2018). Ten simple rules for documenting scientific software. PLOS Computational Biology. 14(12). Retrieved from https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006561.

Mastropasqua, F. (2016). You Might Be an Amateur Programmer and Not Even Know It. Retrieved from https://www.clearlyagileinc.com/agile-blog/2016/5/21/you-might-be-an-amateur-programmer-and-not-even-know-it.

Miles, R., Hamilton, K. (2006). Learning UML 2.0. Switzerland: O'Reilly Media.

Morgan, A., (2017). The Modern Application Stack – Part 1: Introducing the MEAN Stack. Retrieved from: https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack

Node.js. (2020). Node.js. Retrieved from https://github.com/nodejs/node

Prana, G. A. A., Treude, C., Thung, F., Atapattu, T., & Lo, D. (2018). Categorizing the Content of GitHub README Files. Retrieved from https://link.springer.com/article/10.1007/s10664-018-9660-3.

Shah, H., & Soomro, T. (2017) Node.js Challenges in Implementation. Retrieved from https://www.researchgate.net/publication/318310544_Nodejs_Challenges_in_Implementation

Stafford, Gary. (2015) Calling Third-Party HTTP-based RESTful APIs from the MEAN Stack. Retrieved from: https://programmaticponderings.com/tag/mean-stack/

Tüzün, E. & Tekinerdogan, B. (2015). Impact of Experience Curve on ROI in Software Product Line Engineering. Inf. Softw. Technol., vol. 59, no. C, pp. 136–148. Retrieved from https://doi.org/10.1016/j.infsof.2014.09.008

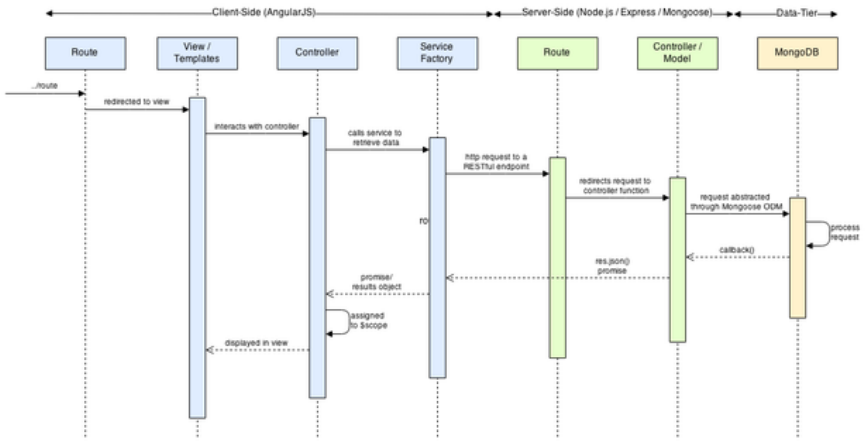
Van Deursen, A., Aniche, M., Aué, J., Slag, R., De Jong, M., Nederlof, A., & Bouwers, E. (2017). A Collaborative Approach to Teaching Software Architecture. *In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17).* Association for Computing Machinery, New York, NY, USA, 591–596. Retrieved from: https://doi-org.proxy.cityu.edu/10.1145/3017680.3017737

Van Loggem, B. (2014). Software Documentation: a Standard for the 21st Century. ISDOC '14: Proceedings of the International Conference on Information Systems and Design of Communication (ISDOC '14). Association for Computing Machinery, New York, NY, USA, 149–154. Retrieved from https://dl-acm-org.proxy.cityu.edu/doi/pdf/10.1145/2618168.2618192

Appendices and Annexures

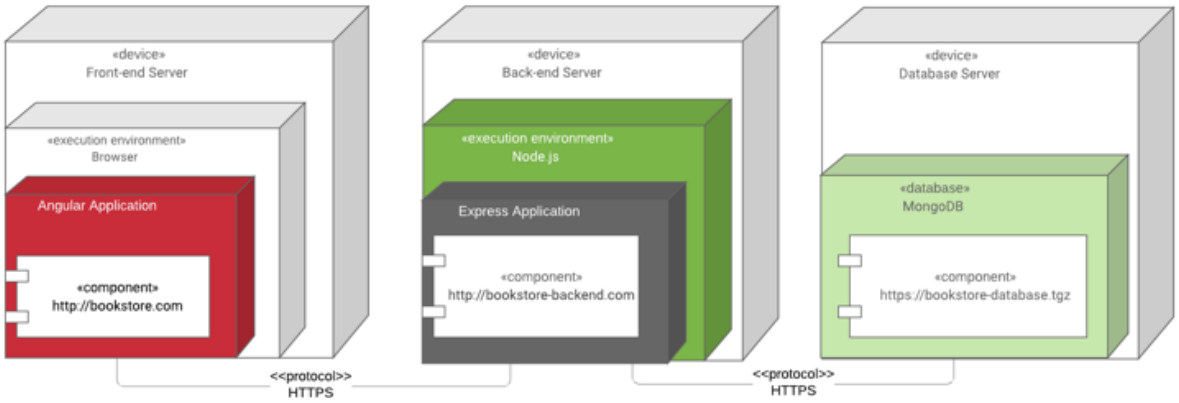
Appendix A

4 + 1 View Sequence Diagram of MEAN Stack Request/Request Data Flow (Stafford, 2015)



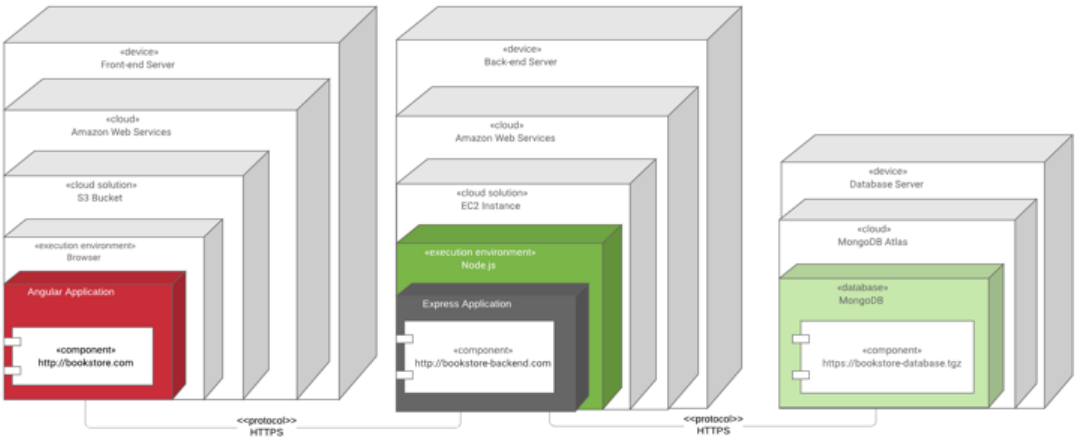
Appendix B

MEAN Stack Deployment View

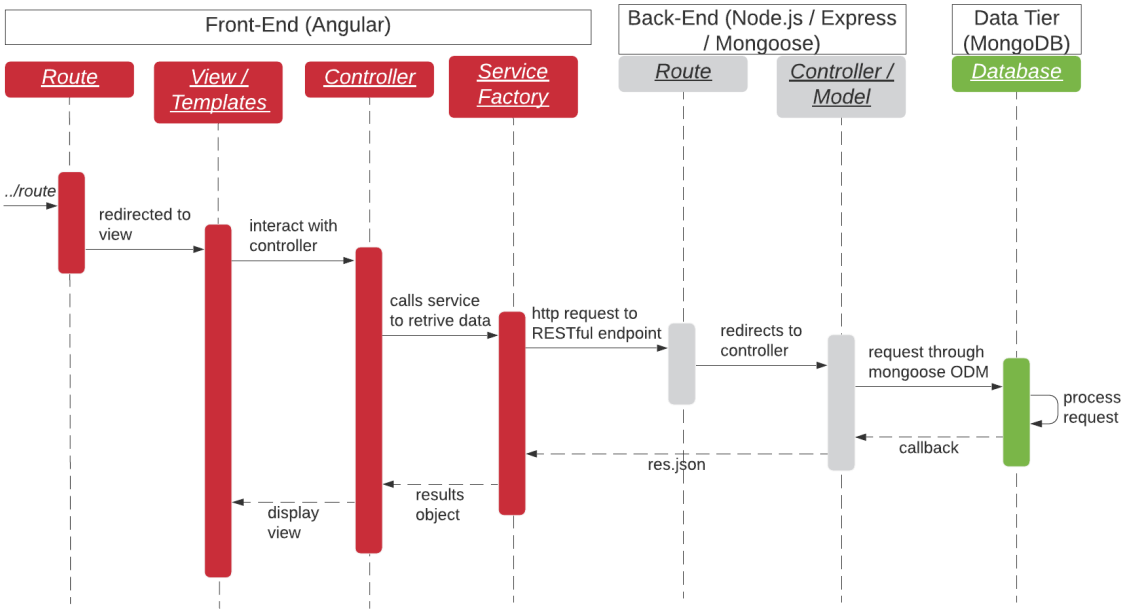


Appendix C

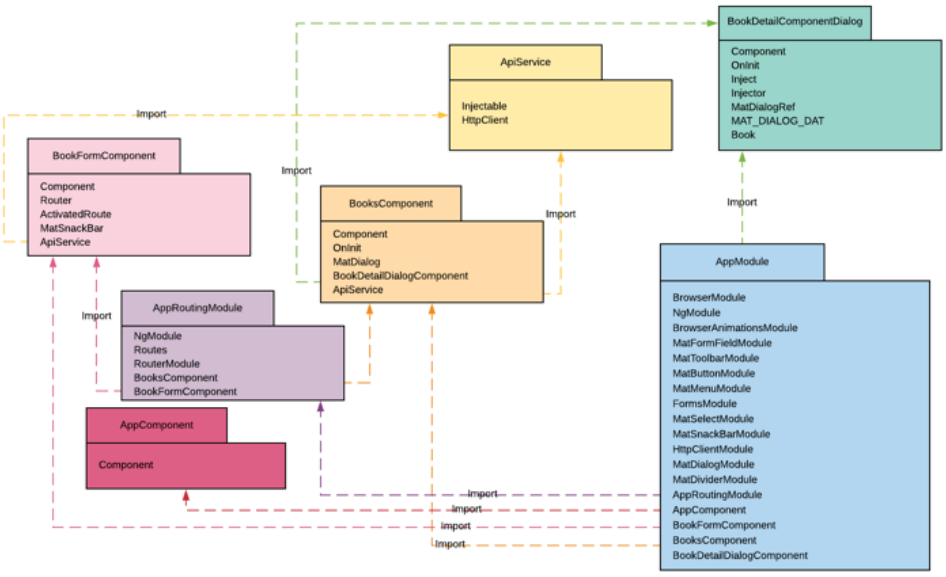
MEAN Stack Deployment View for Production



Appendix D
MEAN Stack Request/Response Data Flow

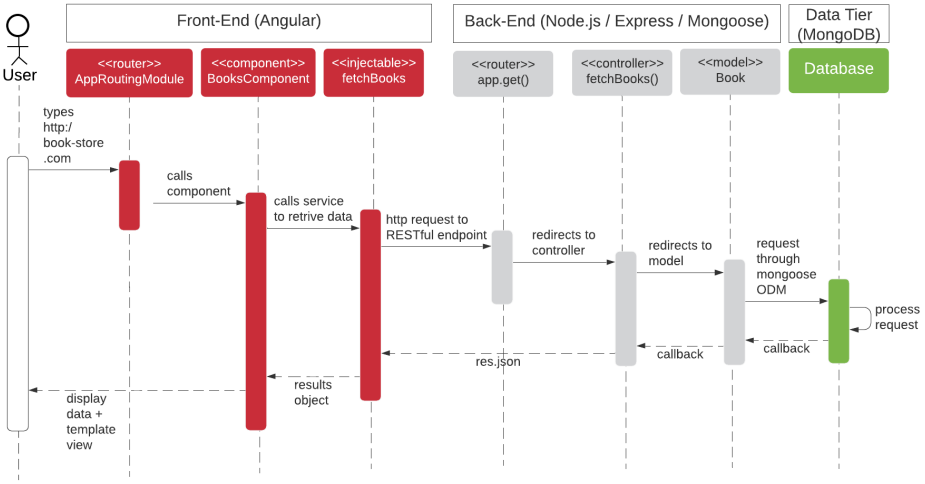


Appendix E
Angular Module Package Diagram



Appendix F

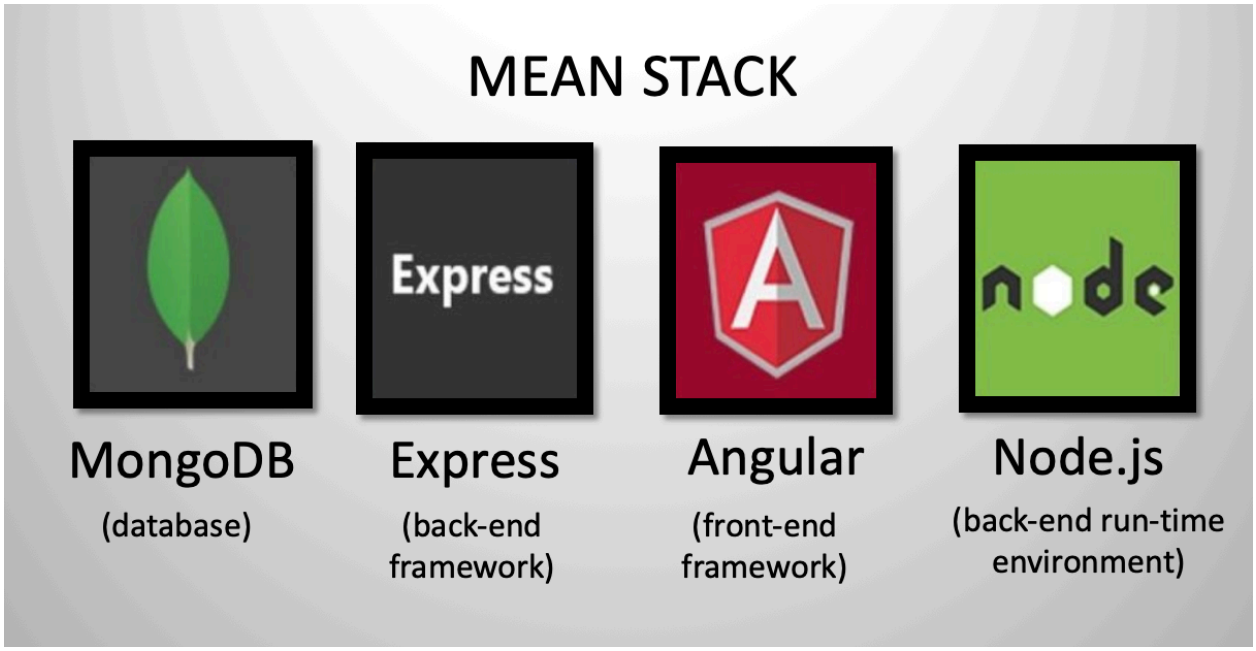
MEAN Stack Scenario using Sequence Diagram



Appendix G

Documentation

MEAN Stack



MEAN Stack is a full-stack JavaScript open-source solution. MEAN Stack consist of MongoDB, Express, Angular, and Node.js. The idea is to solve the common issues with connecting those frameworks, build a robust framework to support daily development needs, and help developers use better practices while working with popular JavaScript components.

Back-end with Node.js

Node.js is built for handling asynchronous I/O while JavaScript is has a event loop built-in for the client-side that makes Node.js fast in performance compared to other environments. However, the event-driven/callback approach makes Node.js difficult to debug and learn as well.

Node.js includes modules such as mongoose, which is a MongoDB object modeling, and express web application framework. Through node modules, abstraction can be achieved, which reduces the overall complexity of the MEAN stack.

Back-end with Express Framework

Express is a minimalist and unopinionated application framework for Node.js. It is a layer on top of Node.js that is feature-rich for web and mobile development without hiding any Node.js functionalities.

Front-end with Angular

Angular is a web development platform built in TypeScript that provides developers with robust tools for creating the client side of web applications. It allows development of single-page web applications where content changes dynamically based on user behavior and preferences. It features dependency injections to ensure whenever a component is changed, other components related to it will be changed automatically.

Database with MongoDB

MongoDB is a NoSQL database which stores data in BJSON (Binary JavaScript Object Notation). MongoDB became the de facto standard database for Node.js applications to fulfill the JavaScript

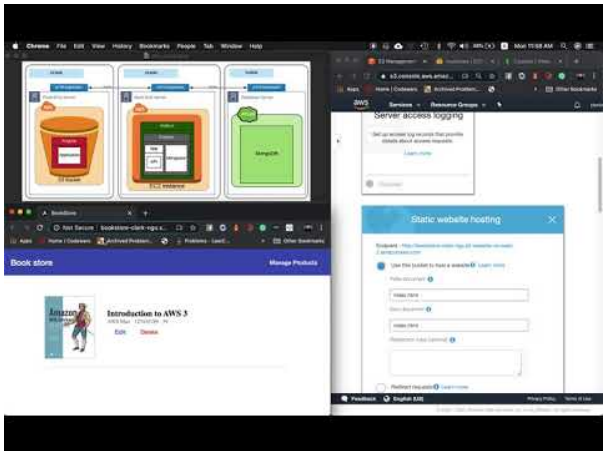
everywhere using JSON (JavaScript Object Notation) to transmit data across different tiers (front-end, back-end, and database).

Physical View using Deployment Diagram

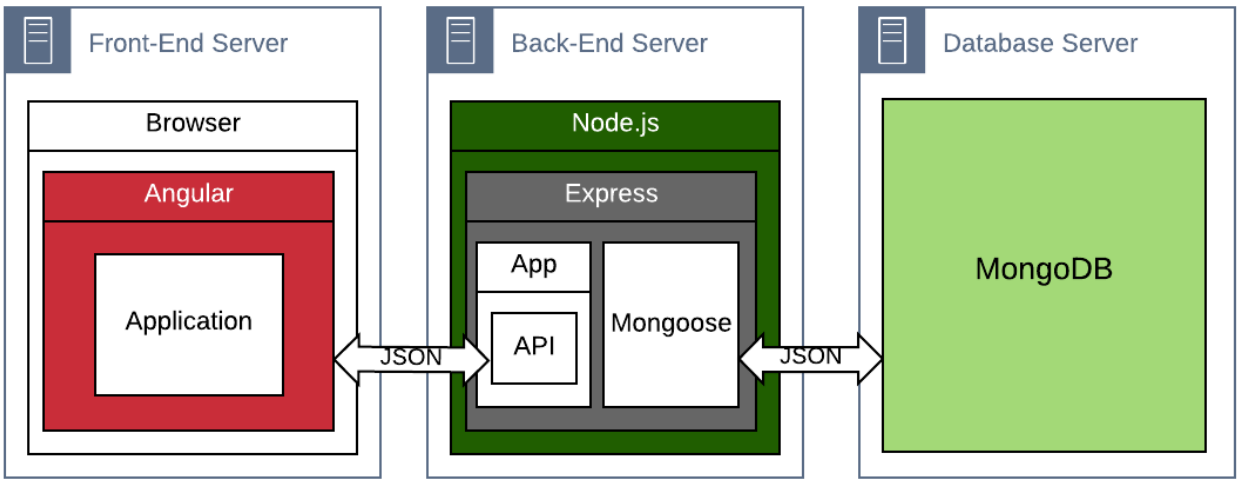
Who uses or what it shows:

- System engineer
- Topology
- Communications

Video Demo

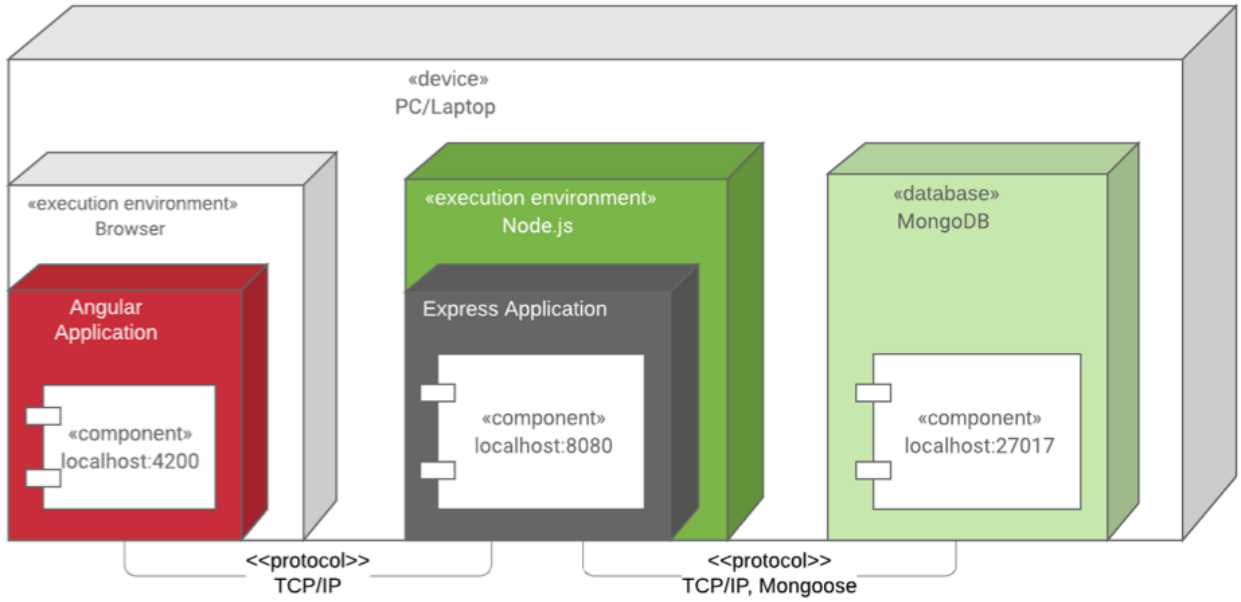


The deployment diagram shows 3 servers: front-end, back-end, and database. In the front-end, we require the browser as angular applications are browser-based web applications. The back-end server hosts our Node.js with Express on top of Node.js. In Express, we have the application and mongoose on top of it. Express will handle the communication on both front-end and database. The database server only includes a MongoDB. JSON is utilized to communicate across servers.

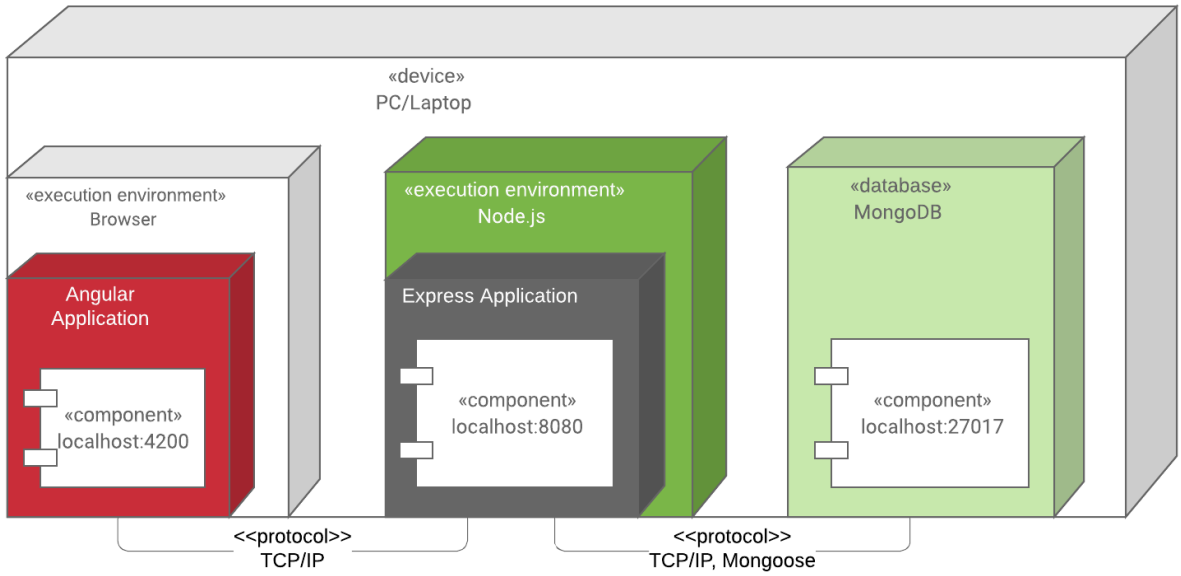


In our first build of MEAN Stack, we'll be deploying locally using our local machine (localhost) to deploy the front-end server, back-end server, and database server. We'll be using the default ports of the following: Angular - port 4200, Node.js/Express – port 3000, and MongoDB – port 27017.

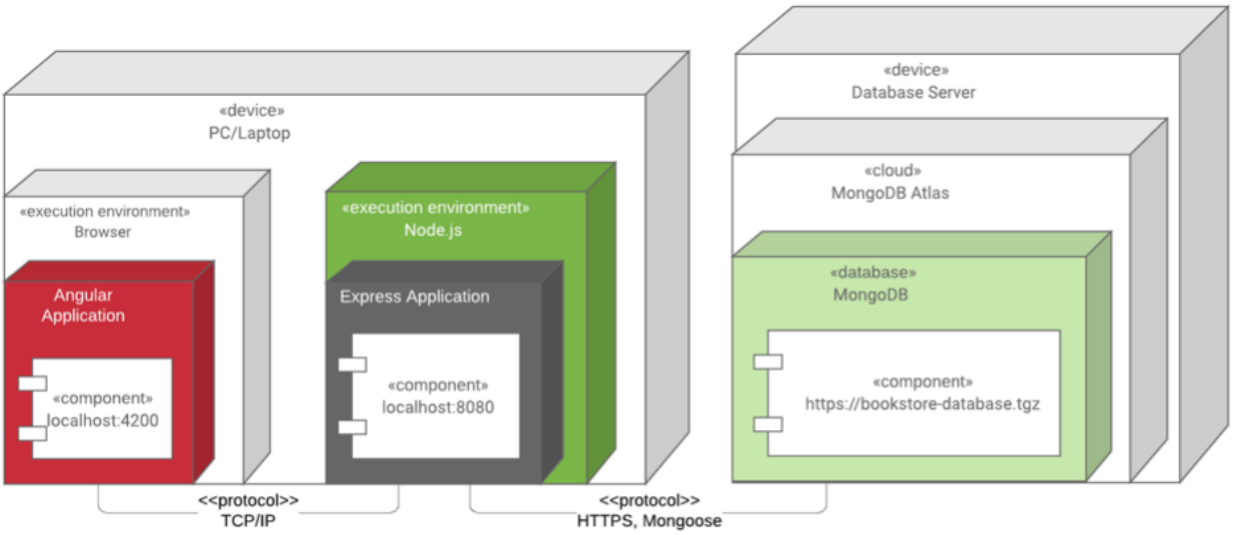
The diagram below shows the full stack web application in UML notation.



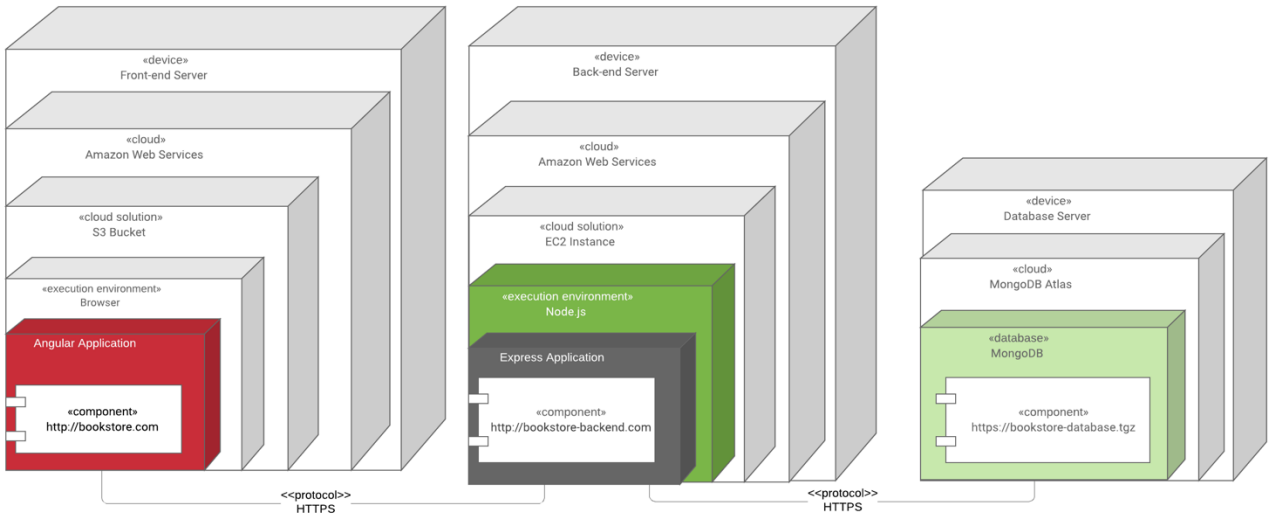
In our first build of MEAN Stack, we'll be deploying locally using our local machine (localhost) to deploy the front-end server, back-end server, and database server. We'll be using the default ports of the following: Angular - port 4200, Node.js/Express – port 3000, and MongoDB – port 27017.



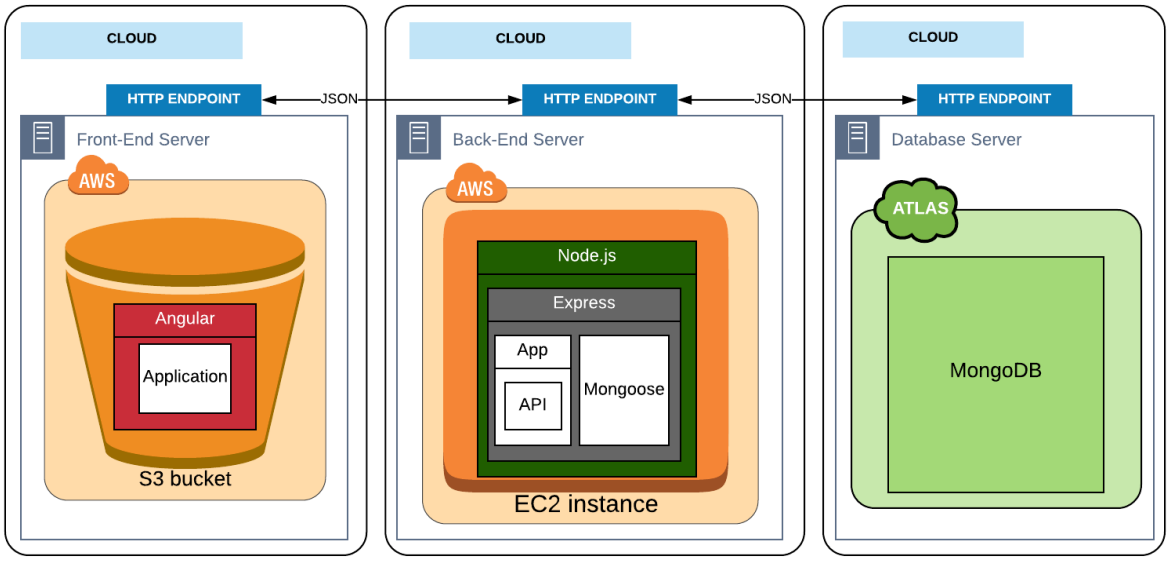
Moving further to actual production, the first step to migrate to the cloud is our database. For MongoDB, MongoDB Atlas was chosen as the cloud solution.



The last step to production deployment is uploading our front-end code to Amazon S3 with AWS, uploading the back-end in an EC2 Instance with AWS. They would all communicate to each other with HTTP endpoints.



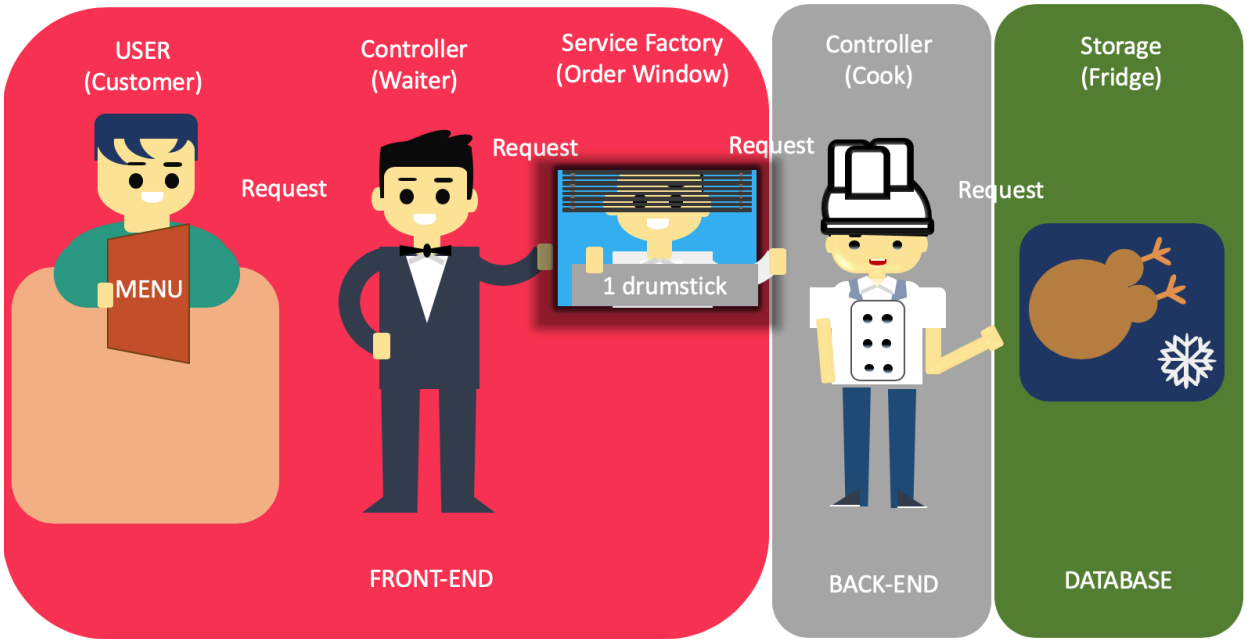
Here's another diagram to show our production deployment without using UML notation.



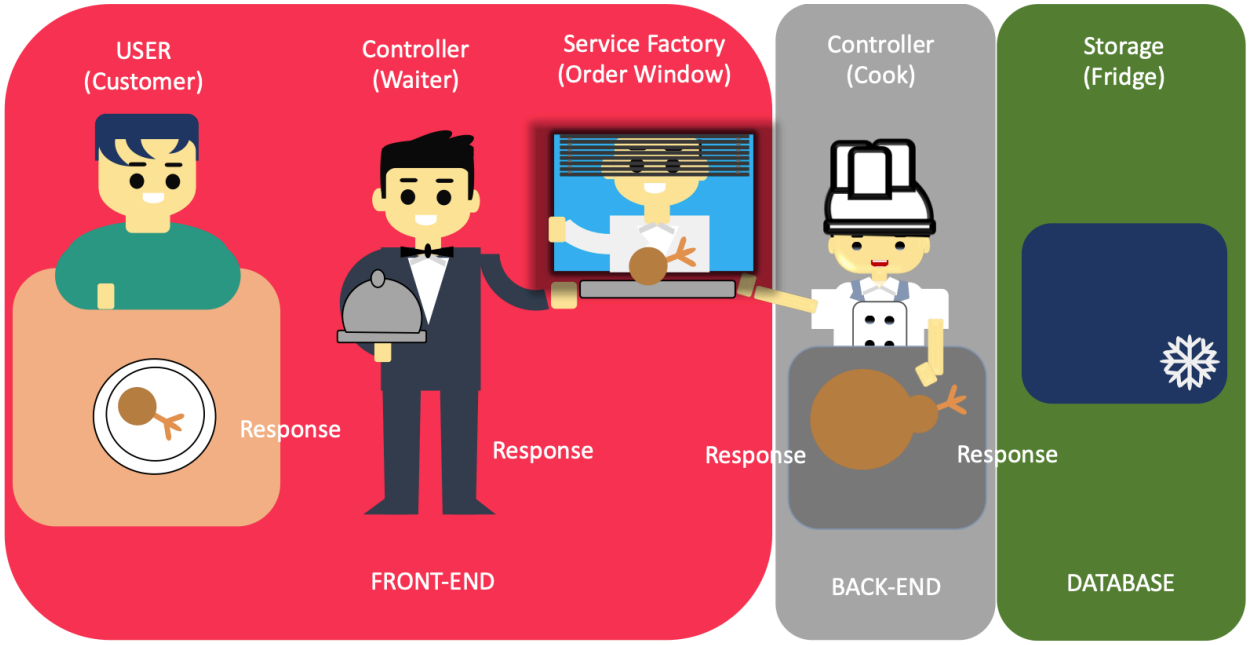
Restaurant Analogy for Live Presentation

As the main topic is about tackling the steep learning curve, a restaurant analogy is shown to let the user understand and retain the process for the request and response process for the full stack application.

The customer (end-user) requests his order through the waiter (controller) and the waiter hands over the request to the person at the order window (service factory). These three components makes up the front-end server. The service factory will be the one to communicate with the cook (controller) in the back-end. The cook will then grab the necessary ingredients (data) in the fridge (database server).



The fridge will be able to provide the necessary material (data) to the cook in the back-end. The cook can now process that data and send back to the service factory of the front-end. The controller (waiter) will hand-over the prepared meal to the customer (user). The customer will now be able to consume the meal (data).

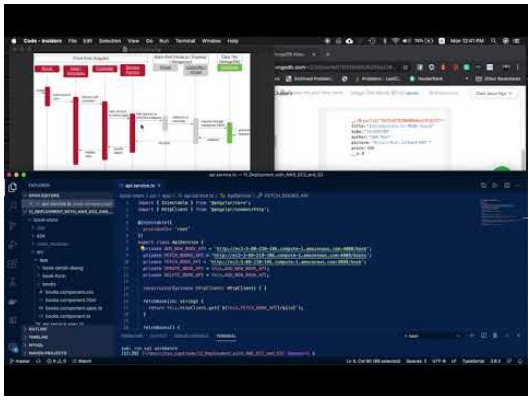


Process View using Sequence Diagram

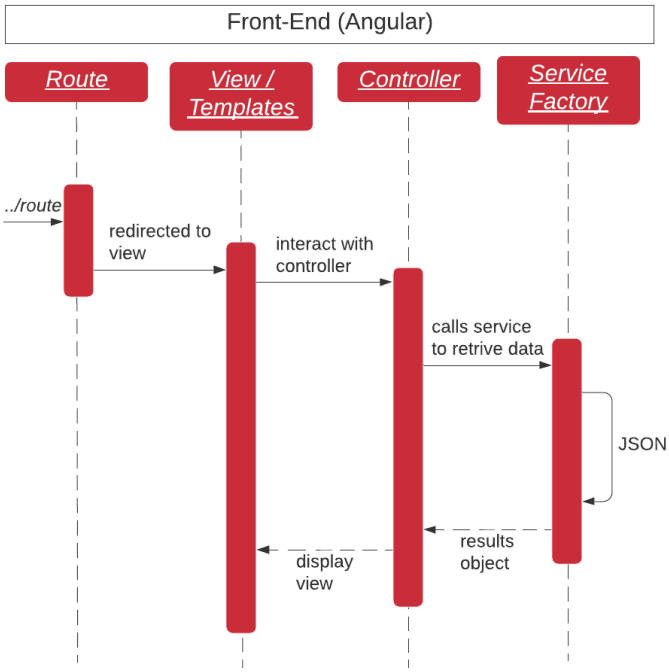
Who uses or what it shows:

- Integrators
- Performance
- Scalability

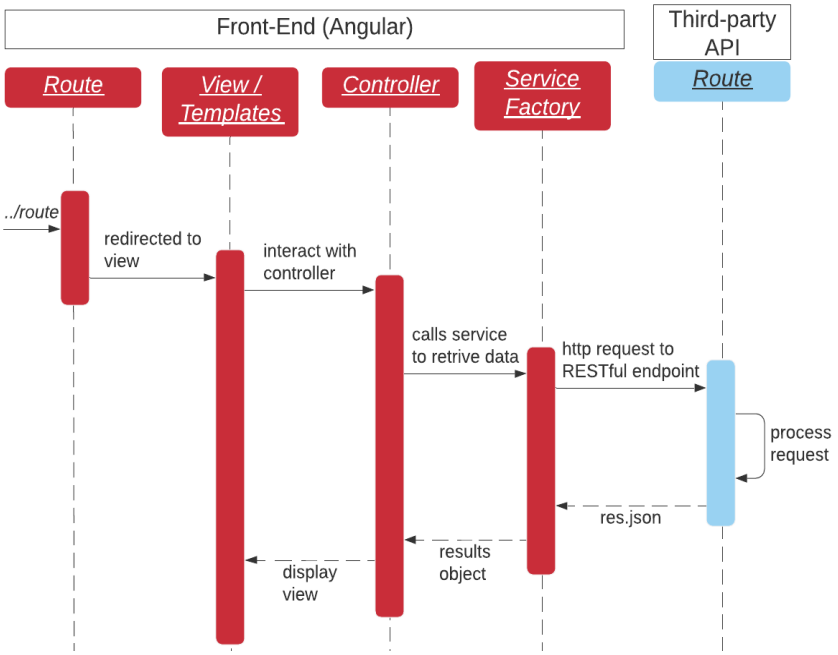
Video Demo



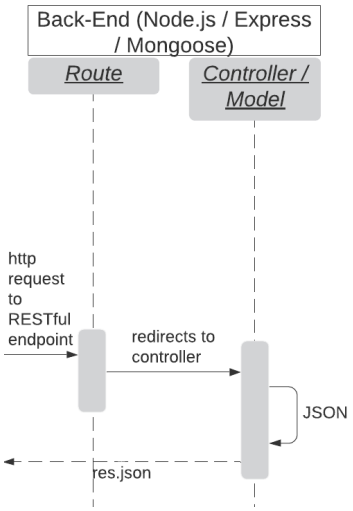
In the process view, the front-end server and back-end server are first shown separately then connect them together with the database server. In the first example, Angular application was deployed with hard-coded JSON in a service.ts file (located in the Service Factory).



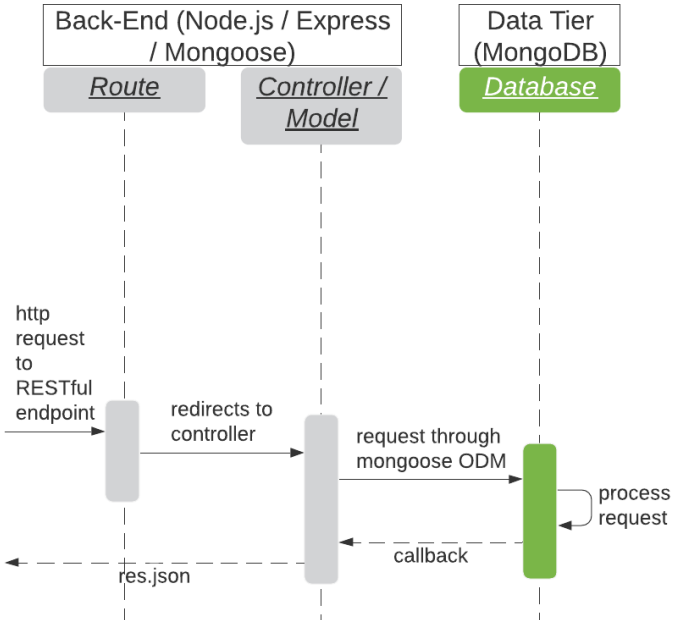
The Angular application can communicate to third-party APIs to grab data and display to the user.



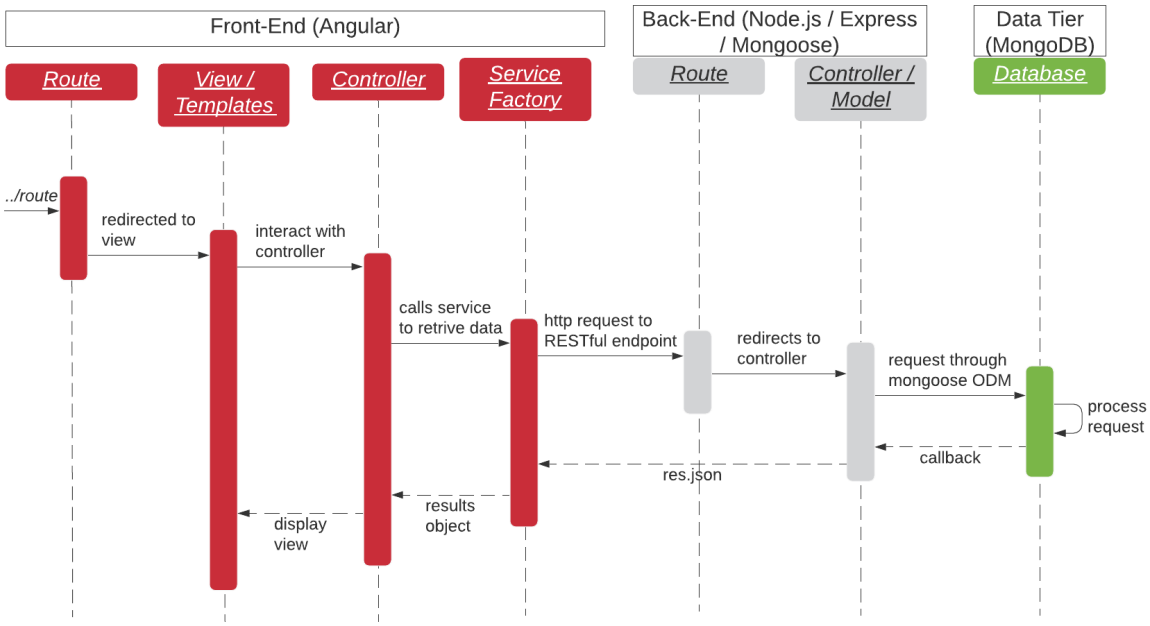
In our back-end, the Node.js application starts with a hard-coded JSON that can be processed and respond with a JSON as well.



This back-end can be connected to third-party APIs or a database server to grab JSON, process, and send back to the requester.



With the front-end server, back-end server, and database server process explained, the combination of these three are shown below:



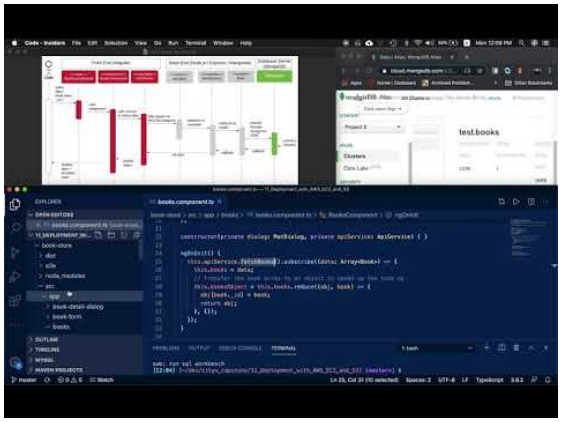
When an http request is made, the front-end will be triggered an Angular will pick up the request. The request will be passed internally in Angular with Route sending a request for the view to View/Template. View/Template will request the Controller. The Controller will then create a http request to a RESTful (Representational state transfer) endpoint to the Server Side, which is Express/Node.js. The request will then be passed internally with Express/Node.js from its Route to the Controller/Model. The Controller/Model will make a request through the Mongoose ODM to interact with the Database Server that has MongoDB. MongoDB will process the request and respond the callback to Express/Node.js. Express/Node.js sends a JSON response to the Angular Controller. Angular Controller would respond with a view.

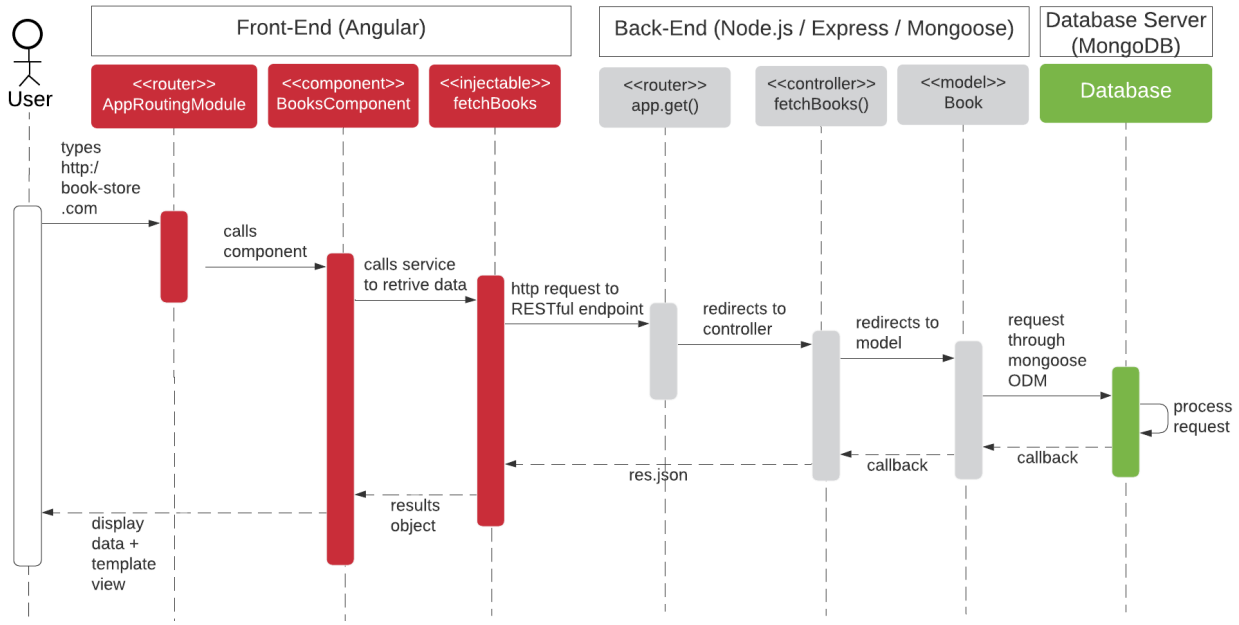
Scenario View using Sequence Diagram

Who uses or what it shows:

- Describe interactions between objects and between processes

Video Demo





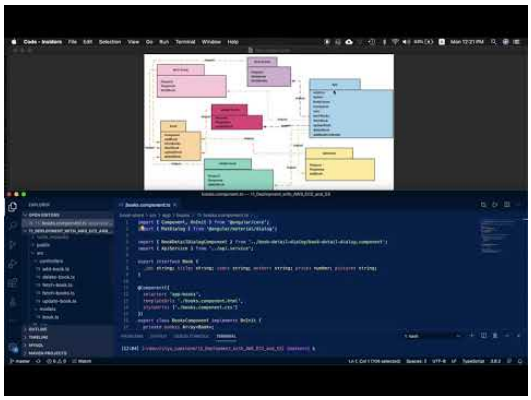
The scenario described is a user accessing a book store application. When the user enters the URL, JavaScript will be run and will hit the router of the front-end server, which is AppRoutingModule. AppRoutingModule will call the BooksComponent, which will load fetchBooks as its dependency injection. fetchBooks will then create an HTTP request to the back-end server that has a router, controller, and model to process the request and query the database server. Database server processes the query and with the back-end server waiting, will grab the data and sent it back to the front-end server as a JSON response. The front-end will now have the data and the template view to show to the user.

Development View using Package Diagram

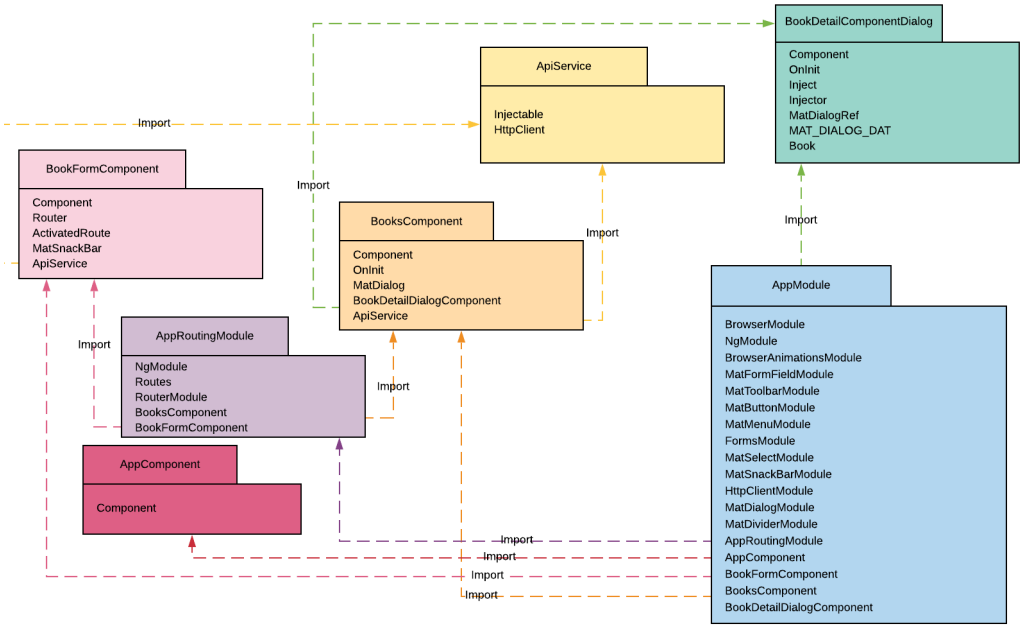
Who uses or what it shows:

- Programmers
- Software Management

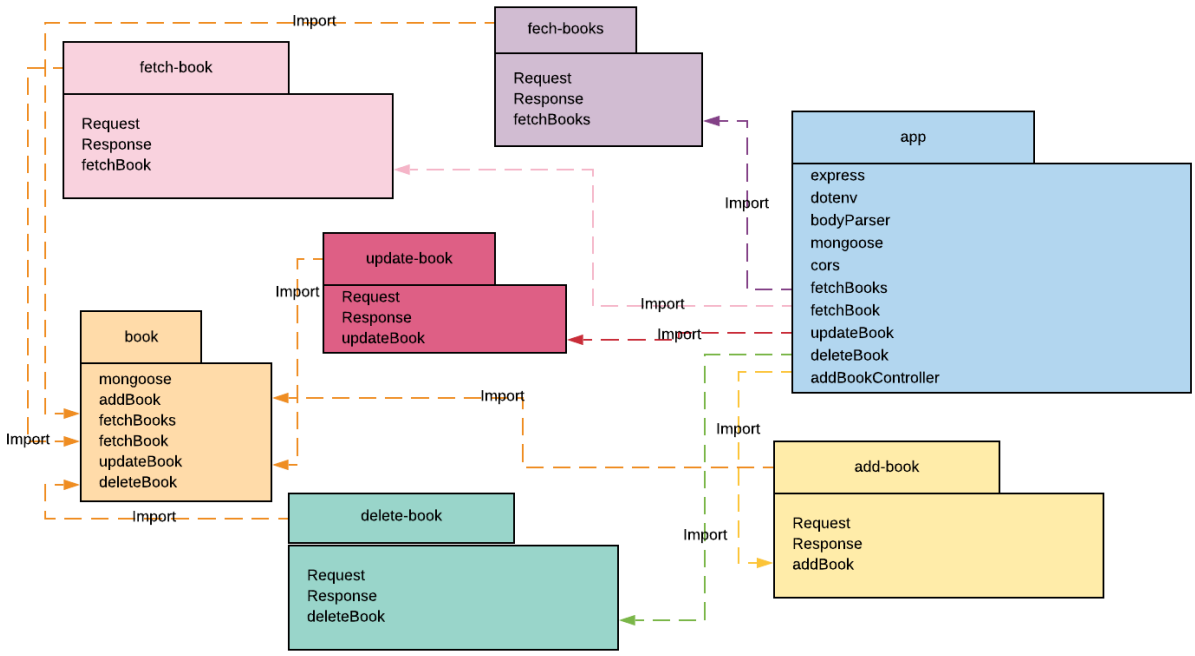
Video Demo



The package view of the Angular application shows that every Angular Component is imported in the AppModule. AppModule and AppRoutingModule is dependent on BooksComponent. The BooksComponent is dependent on BookDetailComponentDialog and ApiService.



The package view of the Node.js application shows that all CRUD operations (controllers) such as fetch all books, fetch a book, update a book, and delete a book are imported by the app. Also, all the CRUD operations logic resides in the model book.

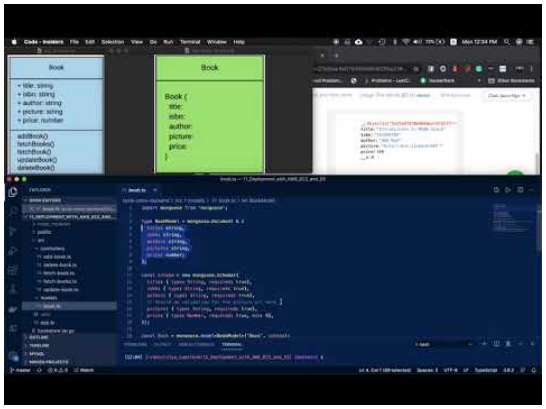


Logical View using Class Diagram

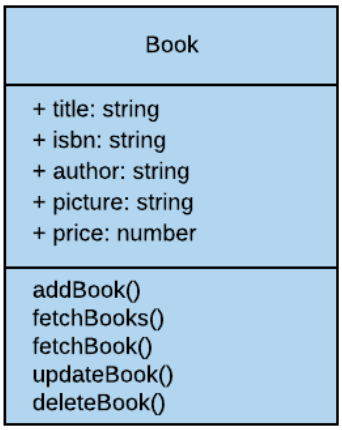
Who uses or what it shows:

- End-user
- Functionality

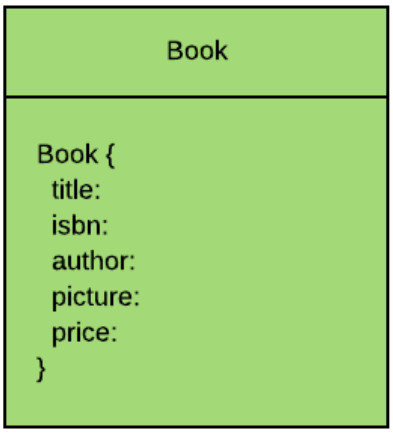
Video Demo



The book store application only showcased a single class called Book. The class members are: title, isbn, author, picture and price. The methods are: addBook, fetchBooks, fetchBook, updateBook, and deleteBook.



The model Book's structure in JSON format.



Appendix H
Survey for Official Documentation

Information about the participant
1. Job Title
Choose one:

- Student

- Employed Tech Role
- Employed Non-tech Role
- Other:

2. Highest Educational Attainment or Currently Studying
Choose one:
- Bachelor - Computer Science Degree
- Bachelor - Non-computer Science Degree
- Masters - Computer Science Degree
- Masters - Non-computer Science Degree
- Other:

3. Years of experiencing with building software (includes building personal projects)
Choose one:
- 0-1
- 2-3
- 4-5
- 5+

Survey for Capstone
This survey consists of two parts: (1) Assessment before checking the documentation, and (2) Assessment after checking the documentation.

Part 1 - Assessment before checking the documentation

	None	Know concepts	Experimented	Built/contributed	Work experience
Documentation	0	1	2	3	4
Architectural Analysis	0	1	2	3	4
Unified Modeling Language	0	1	2	3	4
Full Stack Web Application	0	1	2	3	4
MEAN Stack	0	1	2	3	4
Open Source	0	1	2	3	4

Part 2 - Assessment after checking the documentation.
Please spend 15-minutes to read and understand the documentation below:
http://meanjs.org/docs/0.5.x/#overview

You are free to go anywhere in the documentation

1. Did the documentation increase your understanding
Not really 1 2 3 4 5 Very much

2. Was the description in the documentation helpful?
Not really 1 2 3 4 5 Very much

3. Were the images/diagrams helpful in your understanding?
Not really 1 2 3 4 5 Very much

4. Which diagram was the most helpful in your understanding?
Long answer text

5. Overall rating of the documentation?
Poor 1 2 3 4 5 Excellent

6. Any suggestions or feedback for the documentation? *
Long answer text

Appendix I
Survey for Author's Documentation

Information about the participant
1. Job Title
Choose one:
- Student
- Employed Tech Role
- Employed Non-tech Role
- Other:

2. Highest Educational Attainment or Currently Studying
Choose one:
- Bachelor - Computer Science Degree
- Bachelor - Non-computer Science Degree
- Masters - Computer Science Degree
- Masters - Non-computer Science Degree
- Other:

3. Years of experiencing with building software (includes building personal projects)
Choose one:
- 0-1
- 2-3
- 4-5
- 5+

Survey for Capstone
This survey consists of two parts: (1) Assessment before checking the documentation, and (2) Assessment after checking the documentation.

Part 1 - Assessment before checking the documentation

	None	Know concepts	Experimented	Built/contributed	Work experience
Documentation	0	1	2	3	4
Architectural Analysis	0	1	2	3	4
Unified Modeling Language	0	1	2	3	4
Full Stack Web Application	0	1	2	3	4
MEAN Stack	0	1	2	3	4
Open Source	0	1	2	3	4

Part 2 - Assessment after checking the documentation.
Please spend 15-minutes to read and understand the documentation below:
https://github.com/clarkngo/cityu_capstone/blob/master/README.md

You are free to go anywhere in the documentation

1. Did the documentation increase your understanding
Not really 1 2 3 4 5 Very much

2. Was the description in the documentation helpful?
Not really 1 2 3 4 5 Very much

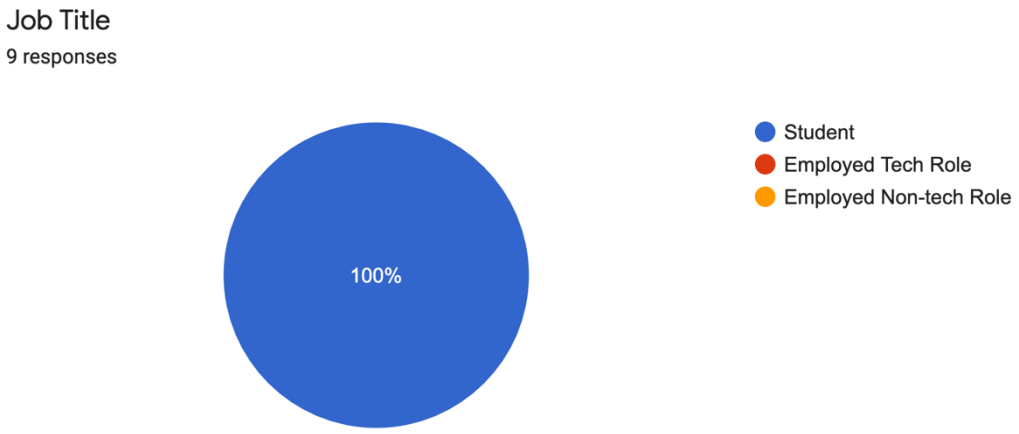
3. Were the images/diagrams helpful in your understanding?
Not really 1 2 3 4 5 Very much

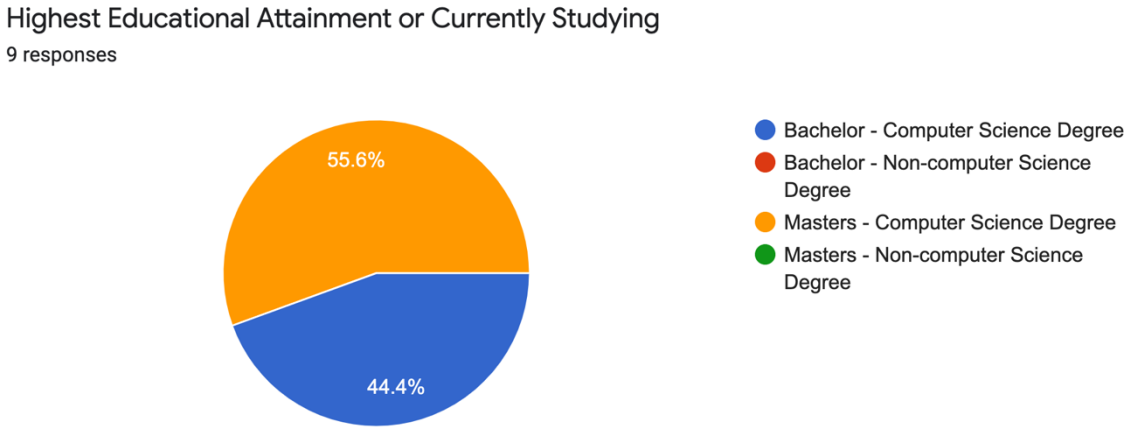
4. Which diagram was the most helpful in your understanding?
Long answer text

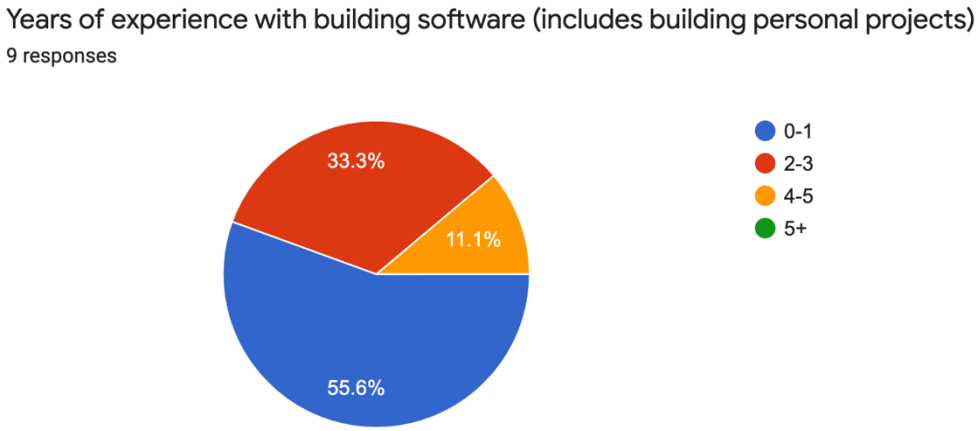
5. Overall rating of the documentation?
Poor 1 2 3 4 5 Excellent

6. Any suggestions or feedback for the documentation? *
Long answer text

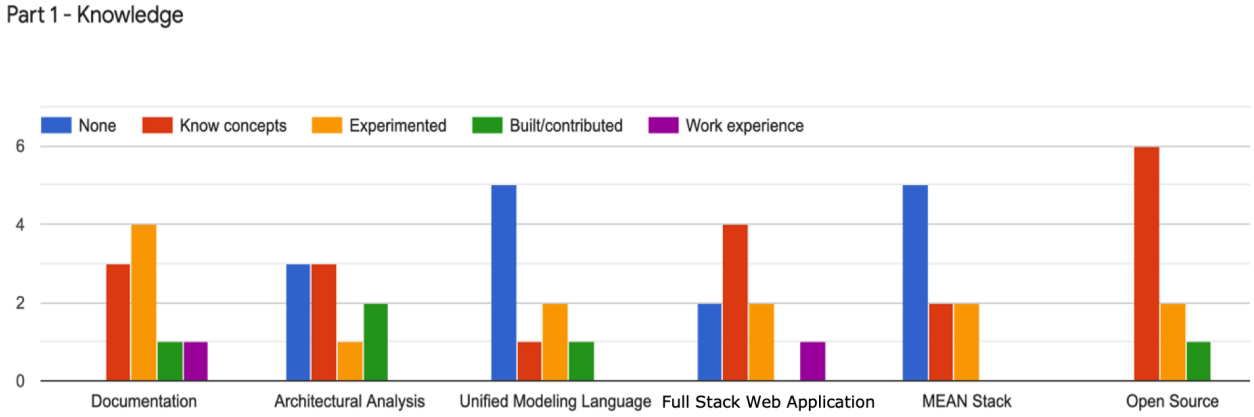
Appendix J
Survey Results for Official Documentation



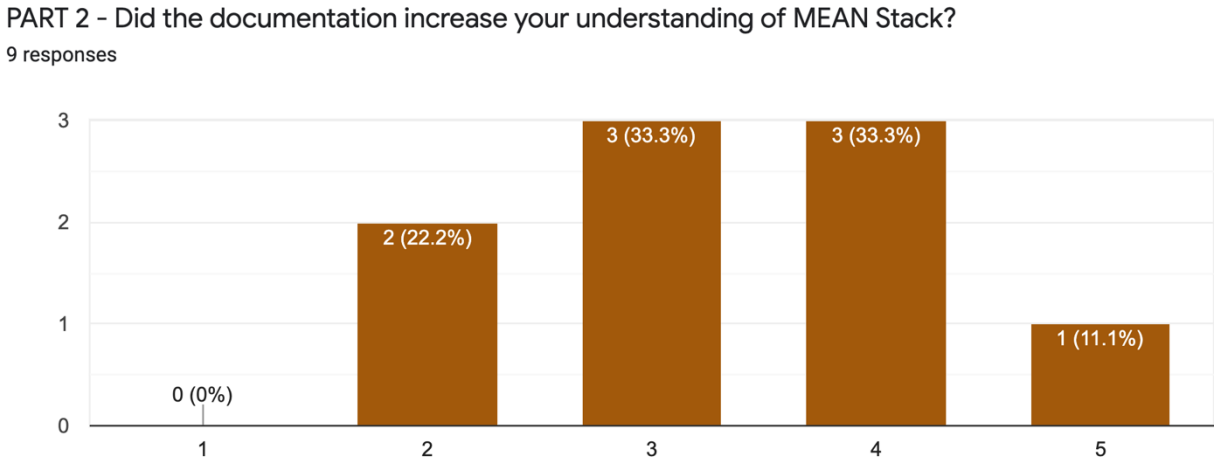




Part 1 - Assessment before checking the documentation

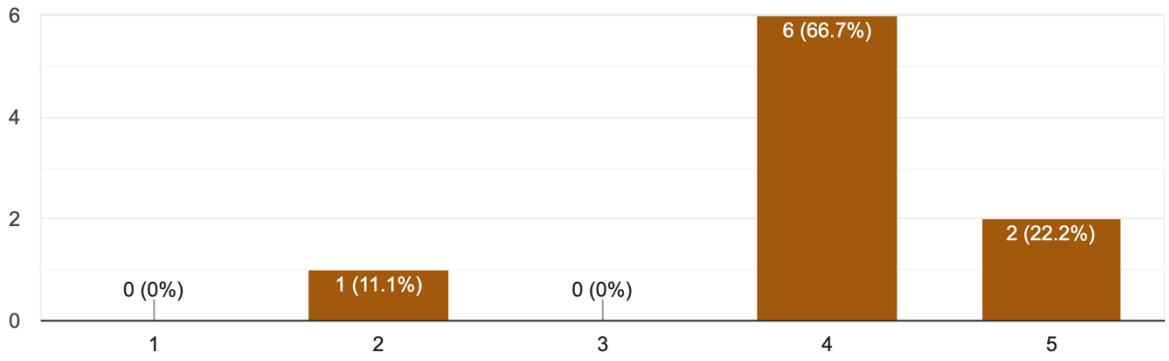


Part 2 - Assessment after checking the documentation.



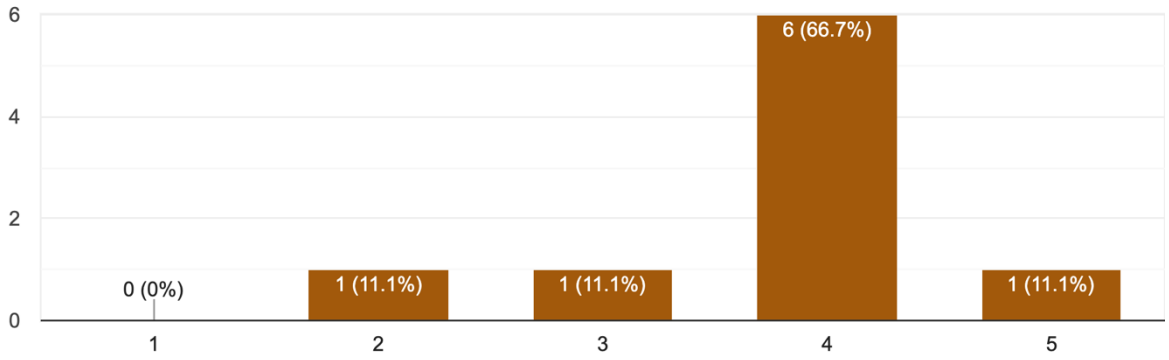
PART 2 - Was the description in the documentation helpful?

9 responses



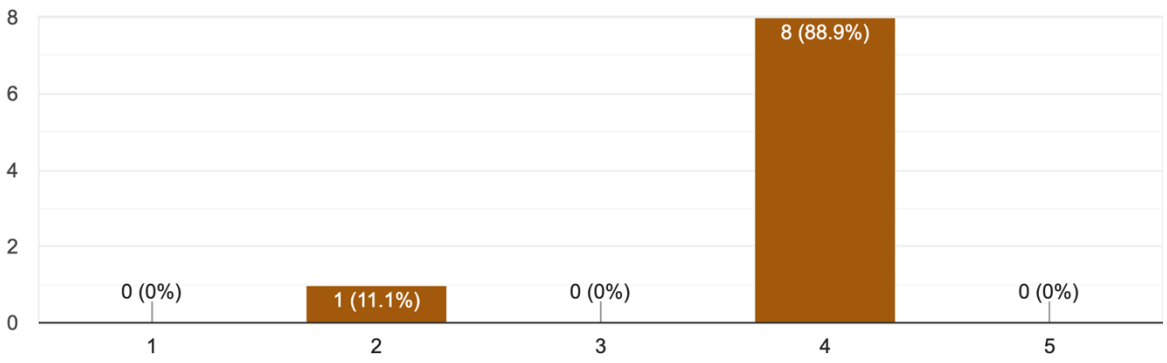
PART 2 - Were the images/diagrams helpful in your understanding?

9 responses



PART 2 - Overall rating of the documentation?

9 responses



Part 2 - Which diagram was the most helpful in your understanding?

9 responses

folders
part1
Routing
Routing image
Breaking down routing
Cmd line examples
ROUTING DIAGRAM
Routign one.

PART 2 - What parts of the documentation were most useful or valuable to you?

9 responses

server test
part2
Mean
Short description on the topics
The CLI prompts for installation and work through
The command examples
ROUTING
Troubleshooting

PART 2 - Any suggestions or feedback for the documentation?

9 responses

nope
no

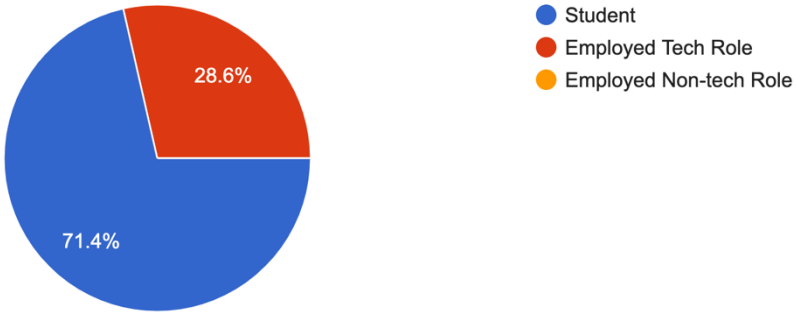
More images, better intro, explain the concept first.
Showing how the front end and back end connectivity takes place would have been better.
None
More pictures. Shorter tech sections. More engaging color scheme. Design for users not for developers.
MORE DETAILS THAT EXPLAIN ANGULAR JS BETTER
Can make it like Mendix academic. Then people know steps order.

Appendix K

Survey Results for Author's Documentation

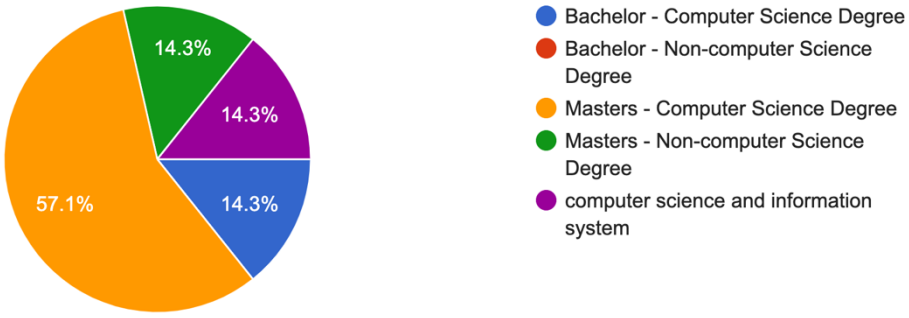
Job Title

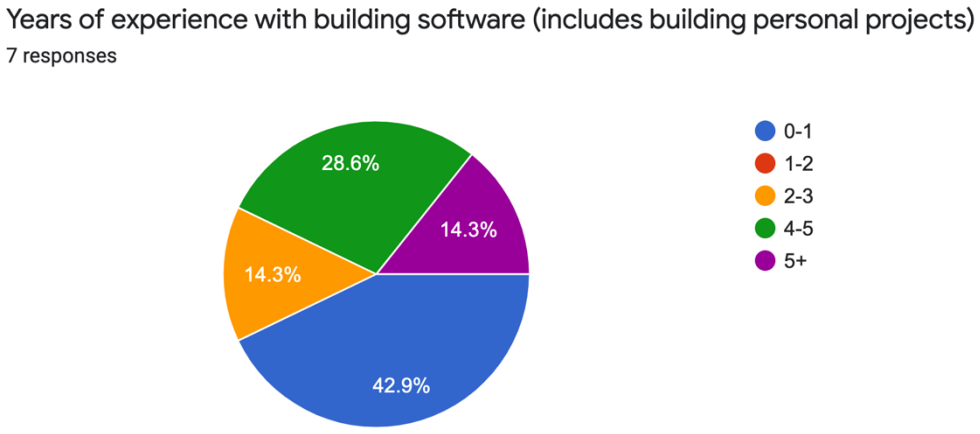
7 responses



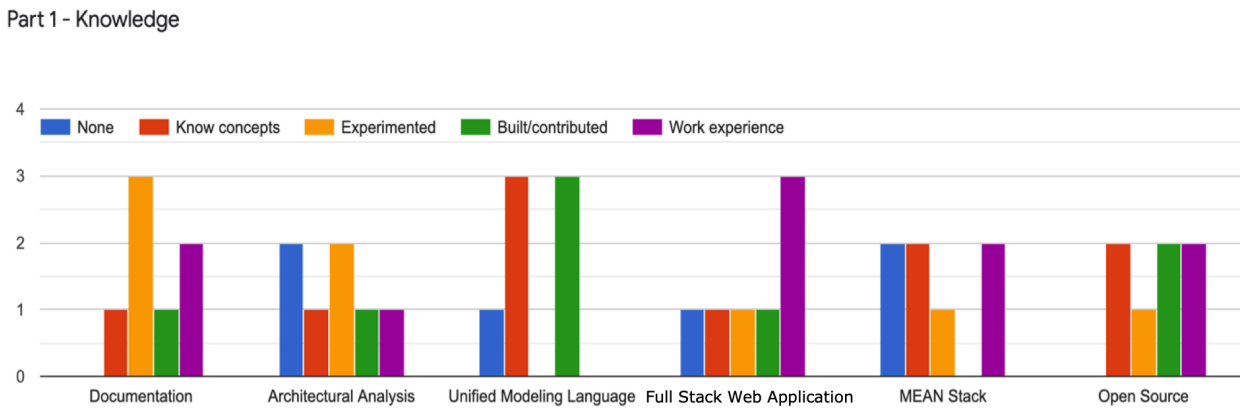
Highest Educational Attainment or Currently Studying

7 responses

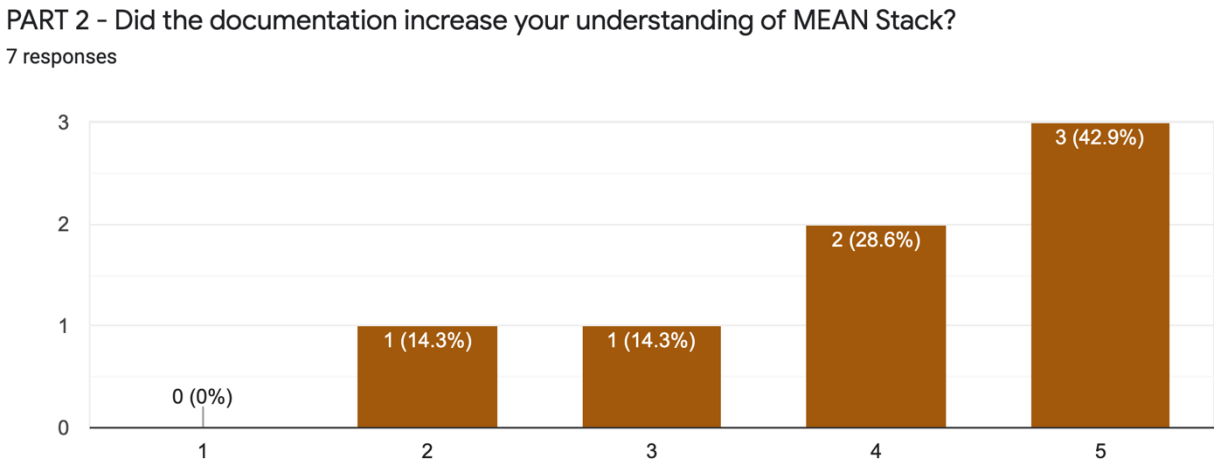




Part 1 - Assessment before checking the documentation

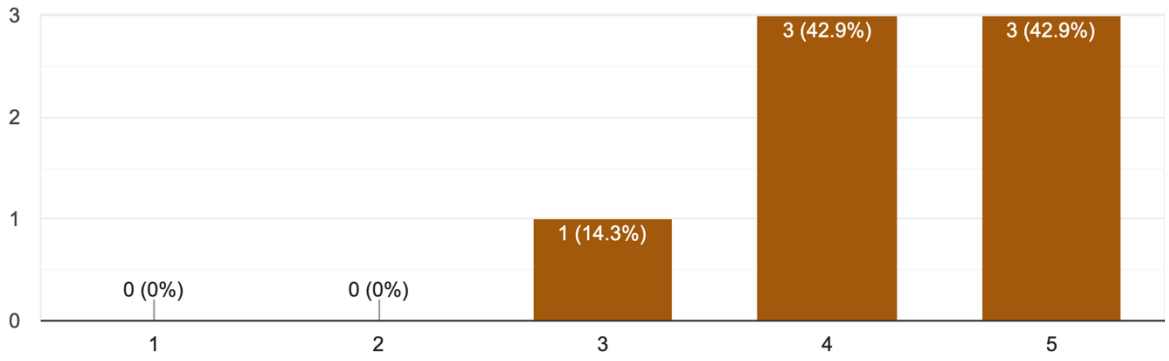


Part 2 - Assessment after checking the documentation



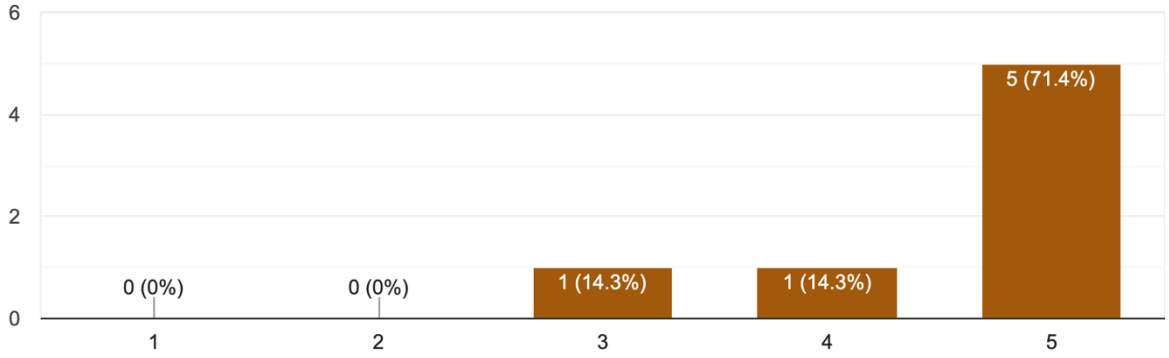
PART 2 - Was the description in the documentation helpful?

7 responses



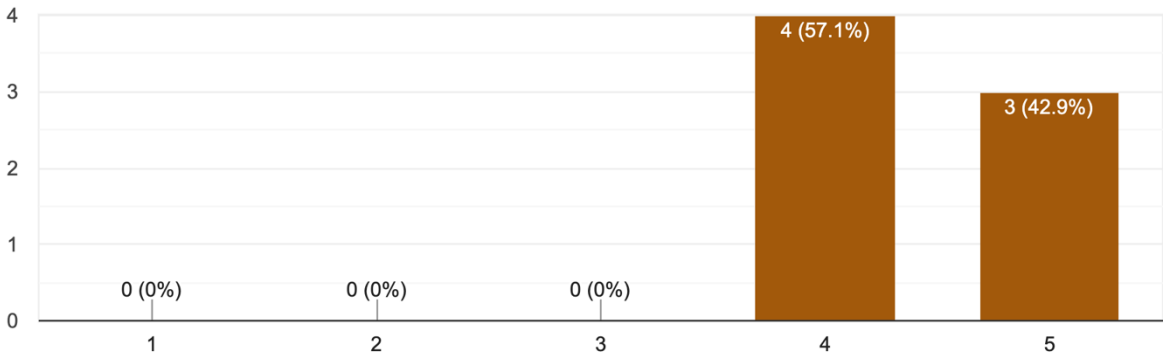
PART 2 - Were the images/diagrams helpful in your understanding?

7 responses



PART 2 - Overall rating of the documentation?

7 responses



Part 2 - Which diagram was the most helpful in your understanding?

7 responses

The one about Express
The one before "retaurant analogy"
Scenarios using Sequence Diagram
Restaurant Analogy
UML notation and restaurant analogy
Physical view using development diagram
Restaurant Analogy and Process View using Sequence Diagram

PART 2 - What parts of the documentation were most useful or valuable to you?

7 responses

Process View using Sequence Diagram
The one talking about TCP/IP
Since i am familiar with dynamic web programming with java, i appreciated you explaining how http request and responses are actually handled.
Part showing how each process works
Images are easy to understand for non-techknowledge person
Development View using Package Diagram. Because I wanted to see the code after looking at the documentation.

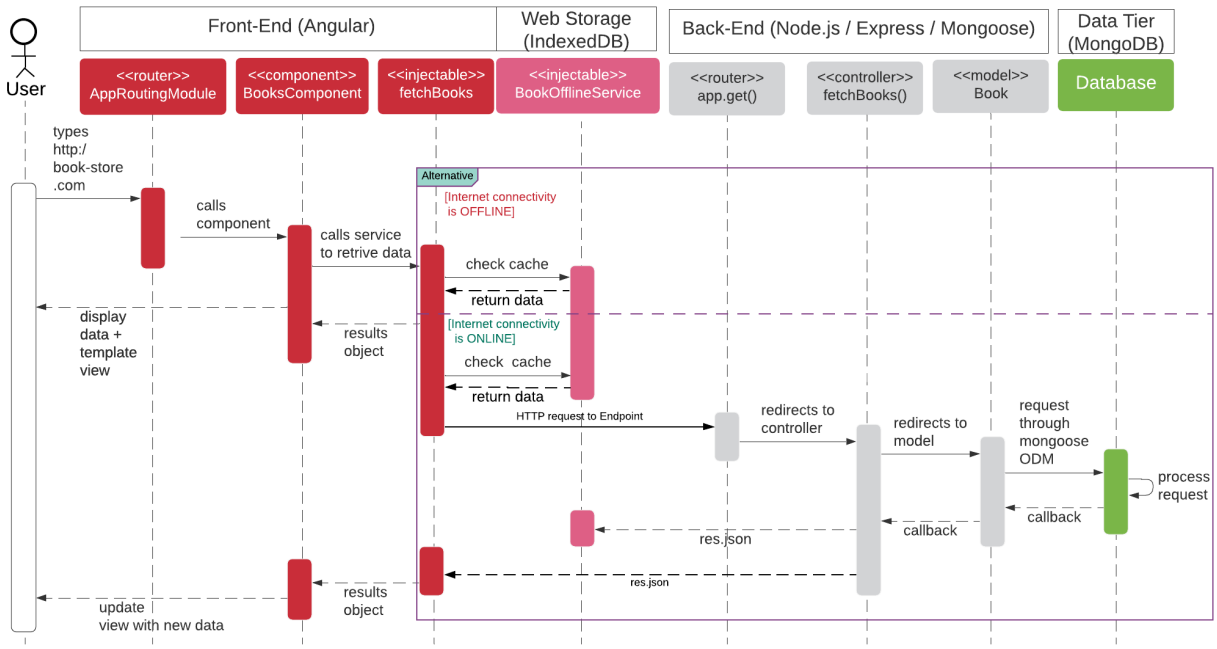
PART 2 - Any suggestions or feedback for the documentation?

7 responses

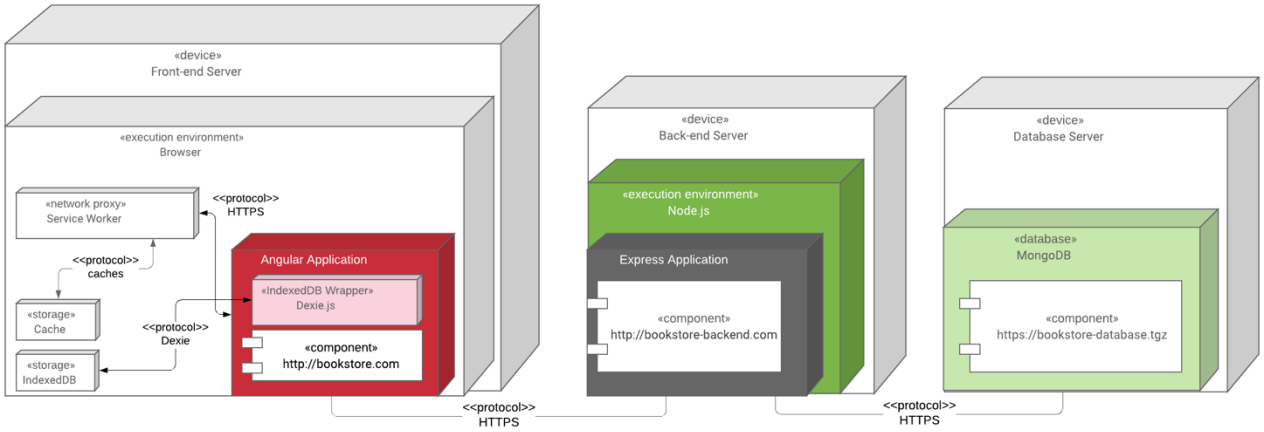
Maybe add a table of contents?
no
It can add more examples for the document.

More description would be amazing
Due to time limit, I had to skim the documentation, so hard to say about the detailed improvements. But perhaps trying explaining with video ? Some people prefer watching than reading.
Visualized explanation is very good for people who don't have enough knowledge.
1. not many people really understand how powerful or what express' purpose really is. you can add some more on that - this will help enrich the docs more 2. this restaurant analogy is really good. one comment is the cook and fridge sometimes can optionally have a service in between them (which can be the "cook/ chef slaves") - just fyi, not required to add it. maybe optional info. 3. "They would all communicate to each other with HTTP endpoints." but the diagram shows https. You should say "with HTTP / HTTPS endpoints" - http and https use different ports and https require certs (they are not the same) 4. front-end is good enough. don't need to say front-end server

Appendix J
Progressive Web Application Scenario View using Sequence Diagram



Appendix L
Progressive Web Application Physical View using Deployment Diagram


«device»
Front-end Server
«execution environment»
Browser
«network proxy»
Service Worker
<<protocol>>
HTTPS
<<protocol>>
caches
«storage»
Cache
«storage»
IndexedDB
<<protocol>>
Dexie
Angular Application
«IndexedDB Wrapper»
Dexie.js
«component»
http://bookstore.com
«device»
Back-end Server
«execution environment»
Node.js
Express Application
«component»
http://bookstore-backend.com
«device»
Database Server
«database»
MongoDB
«component»
https://bookstore-database.tgz
<<protocol>>
HTTPS
<<protocol>>
HTTPS