**Software Design Document**

**Book Store**

**Informatics Engineering**

**PAW - Programming in Web Environment**

**2nd Year, 2nd Semester**

**Last Revision: 11 June 2022**

**Version: 2.0**

**Index**

[**Introduction**](#_ip3f4eq4w1jx) **3**

[**Business Requirements**](#_ajmcz4xef9pu) **4**

[Functional Requirements](#_g90gvx44oger) 4

[1st Milestone](#_gc5vdq57zx1v) 4

[2nd Milestone](#_xdlaii4mqezc) 4

[**Software Specification**](#_8no6vwppsjc1) **6**

[Use Case Design](#_z6mk989s7mgd) 6

[Component Design](#_2qjt817i93k7) 7

[Data Design](#_xpvj1sn4mi8k) 8

[**In-Depth Architectural Design**](#_irry7b4l5u7o) **9**

[Technological Stack](#_ncsorxijvn4r) 9

[Software Architecture](#_yf6dbypt414w) 10

[Software Design Considerations](#_3wextnz0w94z) 10

[**Development Report**](#_pi4zc9egkasq) **11**

[**Business Report**](#_d7048ghcz76) **12**

# 

# Introduction

The purpose of this document is to describe the implementation of the BookStore API described in the Business Requirements. As a secondary purpose this document is written to obtain a distinct note and approval to the curricular unit in continuous evaluation.

The basic architecture is a webstore from a client-server paradigm. The software will consist of two major components. First, the REST API is built on top of the MVC paradigm which will serve as a way of communication with the storefront using requests and responses over the HTTP protocol, and the second as a storefront.

The API offers a full range of e-commerce options making it possible for customers to view books and sell books, add books to a cart, and check out. For store owners, the platform includes a management dashboard to manage books, purchases and employees.

The expected audience is the professors from PAW curricular unit, the professor from the theoretical classes FAS, and the professors from the practical classes JRMR, PJS, MFG. It also serves as a reference for students.

This document includes but is not limited to, an Architectural Overview, Design and Implementation decisions and Functional Requirements.

This document was adapted from the IEEE Recommended Practice for Software Design Documents. Some of the IEEE Recommended Practice for Software Design Documents have been reduced in order to simplify this assignment.

# Business Requirements

A small company in the library business started a project to modernize and digitize its business. As a niche company, it intends to keep its DNA based on customer service by selling book-shaped works of art in its various editions. To give a complete response to the assignment, we decided to build a storefront that will accommodate several e-commerce features on behalf of the customer, as well as the librarian employees.

In this section, we will discuss and document all functional requirements for each milestone to ensure that any requested functionality is attended and we will be updating this document to maintain that information up-to-date.

## Functional Requirements

Below he describes the functional requirements planned for each milestone:

### 1st Milestone

* The employees need to be able to manage the books, whether new or used, in the warehouse (view, update information and its quantity, create and delete).
* The employees need to be able to register a purchase on-site, whether the client has an account or not, so they can retain customer purchase history. enhancement
* The employees need to be able to register a client so they can manage the client fidelity program.
* The company owner needs to be able to manage their employees (view, update information, create and delete)

### 2nd Milestone

* As an online customer, I need to search for books, so that I can find the ones I want to buy
* As an online customer, I must be able to write reviews and rate books so both the store owner and other customers can have a feedback
* As an online customer, I must be able to sign up the newsletter so I can receive discounts and new launches
* As a bookseller, I must be able to sell books so I can make profit
* As a bookseller, I must have a sales dashboard so I can measure my performance
* As a customer I must be able to sign up and sign in on the store so I can buy books

# Software Specification

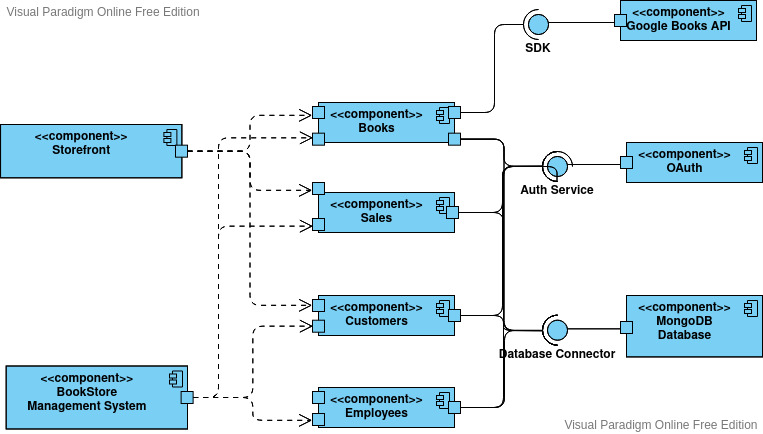
This section describes the basic functionalities of the following modules: Books, Sales, Customers, Employees, Auth. What do the modules represent and what data structures, as well as their functionalities.

## Use Case Design

## 

## 

## Component Design



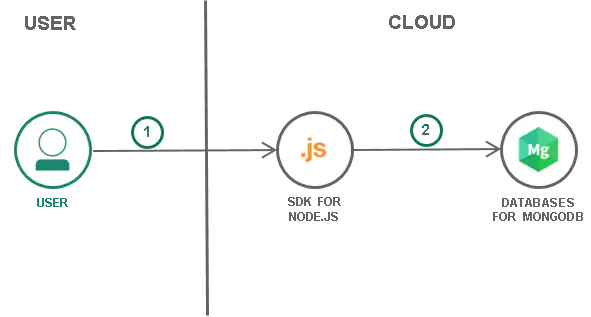
## Data Design

## 

# In-Depth Architectural Design

## Technological Stack

This project uses a MEAN stack (MongoDB, Express.js, Angular and Node.js), allowing us to build a 3-tier application (back-end, front-end, database). Express and Node.Js make up the middle tier handling all communication between the front-end and the database. Express.js is the server-side web framework chosen to build the API along with Javascript. In the top-tier of the stack, for the back-office, as a template engine, we use Embedded JavaScript templates (EJS), when it comes to the storefront we use Angular along with Typescript. MongoDB comes at the lowest level, used to store any data related to customers, books and purchases.



## Software Architecture

Looking at the API folder we can see right away that the project structure follows a layered architecture combined with the MVC structure, where each layer is a folder, we spotlight the following layers:

* Handlers - serves as the controller layer in MVC structure.
* Helpers - serves as decoupled functions used commonly through the project.
* Middlewares - serves as a sandwich layer between routes and handlers.
* Routes - serves as an entrypoint for request defining endpoints and http verbs
* Schemas - serves as database mapper for MongoDB documents allowing specifying these documents to schema definitions. Also serves a model layer in MVC structure.
* Views - serves as view layer in MVC structure

## Software Design Considerations

At first we had considered using a modular structure where each module includes in a folder a handler, schema, route and its validators just like Angular. Express.Js is an un-opinionated minimalistic framework that shows to be flexible but it's easier to diverge from the good practices. As the time was not much, we have chosen to follow the default architecture, the layered architecture. Being mandatory to use the MVC development pattern, we rapidly split each layer into a separate folder and start working. Bellow we considered the key benefits of using a layered architecture versus the cons:

* One of the side effects of creating a layered architecture, is an explosion in directory count and file count, for us: a schema, view, handlers, routes and so on.
* A layered architecture makes sure system components are decoupled, and changes in any of the components of the integrated system do not directly translate into a rewrite of dependent modules.

# Development Report

Perhaps one of the things that got us a slow start was the structuring shift from modular architecture to a layered architecture. This structure transition was justified by the group members because of our lack of experience in modularizing an application and manipulating the framework. We splitted the project into two branches: the master and the next branch, the next branch includes the new structure utilized to build the project.

Another aspect of web development is that developing a REST API with Javascript can be very frustrating especially if you can't find the bugs. Since JavaScript is not a type-safe language it tries to coerce bugs, leading to errors or undesirable behavior due to using an incorrect type. IDE's like Intellij or even a text editor such as vscode are not that good at Javascript too! IntelliSense won't work as expected for Javascript as they work for languages like Java. In the second milestone, this risk of miss typing variables has decreased due to the usage of Typescript to build the storefront in Angular.

As for Angular, the learning curve isrelatively highwhen compared to the simplicity that an MVC structure with ExpressJs possesses. At first, we weren't sure what should be a component or a module. Transferring data across components was the most tricky part, we had to implement a service layer through several modules to fetch data from our API. The service layer also serves as a state transfer object between two components, these components don't need to be necessarily children of a parent component, instead, they can be injected into the constructor of the component classes. Compared to frameworks like React, React provides an easy way to transfer state across components with React Hooks. Overall, we think that Object Oriented Programming is not suitable to build a front-end in the way Angular is implemented.

# Business Report

The innovative aspect of this project lies in the construction of an API and storefront for the library business, using a MEAN stack offers flexibility, consistent language, and scalability for any web development project.

About the daily added value of the library business, they will get several advantages, which stand out:

* Improved management of the stock of the books
* Better organizational management at the employee level.
* Customer rewards for their continuous acquisition at the online store

All these advantages will translate into increased efficiency in sales and management. From another perspective, the present work resulted in obvious benefits for students at various levels:

* The project developed at the bookstore, allowed us to acquire new knowledge that led to opening new horizons and perspectives for those who want to work in tech.
* The technological stack applied in this work helped to better internalize and structure the concepts taught this semester on web programming.