Leopold-Franzens Universität Innsbruck
Department of Business Informatics, Production Management and Logistics

PS Value Adding Processes

Report Value Adding Processes

Theoretical Part of Deterministic Models for Lot Sizing

from

Tamino Gaub	12314484
Maximilian Stablum	12312185

Submission Date:	February 8, 2024
Supervisor:	Assoz.-Prof. Mag. Mag. Stefan Haeussler, PhD

Abstract

The Economic Order Quantity model and the Wagner-Whitin algorithm are two deterministic lot sizing models that provide a foundation for optimizing inventory levels and production schedules, reducing costs, and improving efficiency. This paper presents the development of a web-based application that provides a frontend for inserting data for the Economic Order Quantity and the Wagner-Whitin algorithm. These calculations are then processes in a Java-based Backend and the output is vizualized in the frontend. The application will be beneficial in a business context, aiding companies in making strategic decisions about inventory management, leading to cost savings and improve resource allocations. The source code of the entire project is publicly available on GitHub[1].

[1] https://github.com/maxstablum/DeterministicModelsLotSizing accessed on 08.02.2024

Contents

1. Motivation 6

2. Introduction to Deterministic Lot Sizing Models 6

3. Foundation Models for State-of-the-Art Theories 7

3.1. Economic Order Quantity . . . 7
3.2. Wagner-Whitin . . . 9
3.3. Theoretical Models Prospection . . . 11
3.4. Current Innovations in Lot Sizing Techniques . . . 12

4. Project Development Framework 12

4.1. Methodical Toolbox . . . 13
4.2. GitHub . . . 13
4.3. Java . . . 13
4.4. JUnit Tests . . . 13
4.5. Apache Maven . . . 15
4.6. Spring Initializr . . . 15
4.7. REpresentational State Transfer . . . 16
4.8. React.js . . . 16
4.9. Postman . . . 16
4.10. Swagger . . . 17
4.11. Expected Outcome . . . 18

5. Backend Implementation 18

5.1. Implementing the REpresentational State Transfer Application Programming Interface . . . 18
5.2. Calculation of the Economic Order Quantity . . . 19
5.3. Calculation of the Wagner-Whitin . . . 19

6. Frontend Implementation 20

6.1. Bootstrap Template for the Frontend . . . 20
6.2. Adjustment of the Components . . . 21
6.3. Structure of Service classes . . . 22

7. Challenges Encountered in Application Development 22

7.1. Challenges in Connecting Front- and Backend with REpresentational State Transfer Application Programming Interface . . . 23
7.2. GitHub Merge Conflict . . . 24

8. Lessons Learned within this Project 24

A. Appendix 28

A.1. Economic Order Quantity Controller . . . 28

A.2. Wagner-Whitin Controller 29
A.3. Detailed Calculation of the Economic Order Quantitiy 30
A.4. Detailed Calculation of the Wagner-Whitin Model 31

Acronyms

API	Application Programming Interface	16
EOQ	Economic Order Quantity	7
HTML	Hypertext Markup Language	16
HTTP	Hypertext Transfer Protocol	14
IDE	Integrated Development Environment	13
JSON	JavaScript Object Notation	14
MRP	Materials Requirement Planning	10
MINLP	Mixed-Integer Nonlinear Programming	12
REST	REpresentational State Transfer	16
UI	User Interface	16
URL	Uniform Resource Locator	21
CORS	Cross-Origin Resource Sharing	23

1. Motivation

In today's business environment, characterized by extremely narrow profit margins, increasingly stringent customer expectations for product quality, and the pressure of reduced lead times, the competitive landscape has become more intense than ever [1]. As a result, companies are compelled to seize every opportunity to enhance and optimize their business processes.

Deterministic models for lot sizing play a crucial role in production and inventory management, particularly in environments where demand can be accurately predicted [2]. These models are essential for optimizing production schedules, minimizing costs, and ensuring timely fulfilment of customer orders. By optimizing lot sizes, companies can reduce excess stock and associated holding costs, thereby enhancing overall operational efficiency. The models are a crucial subject for modern businesses as they continue to grapple with the challenge of determining the ideal lot size. This topic is particularly relevant in today's competitive market, where effective resource allocation and cost management are key to achieving business success. The decision problem consists of finding the best trade-off between setup costs and holding costs.

2. Introduction to Deterministic Lot Sizing Models

In deterministic models, demand is assumed to be predictable and constant throughout the planning horizon, and must be fulfilled either through immediate orders or existing inventory [2]. These models specifically address the coordinated lot-size problem, which involves determining a timetable for replenishment [2, 3]. The mathematical complexity of the coordinated lot sizing problem is NP-complete, suggesting that it is unlikely that a polynomial bound algorithm can be found for its solution. For this reason, an extensive literature base is rapidly developing that describes alternative mathematical formulations and exact solution approaches for the problem to solve large industrial problems that can involve over a hundred items and time periods.

The models provide a foundation for optimizing inventory levels and production schedules, reducing costs, and improving efficiency. In particular, the coordinated lot sizing problem exemplifies the complexity and practical challenges faced in this field. The problem's NP-completeness underscores the difficulty in finding efficient, exact solutions, driving research towards innovative mathematical models and solution methods.

Despite these advances, there are remaining significant challenges and opportunities for future research in deterministic lot sizing. The complexity of the problem necessitates ongoing efforts to develop more efficient and effective solution methods. These include both exact algorithms, which guarantee optimal solutions, and heuristic approaches, which provide good solutions in a reasonable timeframe.

Furthermore, the integration of deterministic models with other planning tools and technologies, such as demand forecasting and real-time data analytics, presents a promising avenue for enhancing their applicability and effectiveness in dynamic business environments.

3. Foundation Models for State-of-the-Art Theories

Production planning and scheduling pose significant challenges for management, constituting a hierarchical process encompassing long-term, medium-term, and short-term decisions [4]. The primary focus of addressing this category of issues lies in determining production lots for multiple items across a planning horizon, whether finite or infinite [5]. The objective is to minimize setup costs, inventory holding costs, production costs, and backlogging costs, all while ensuring the fulfilment of known demand. Notably, there is an absence of interdependency between items, owing to the lack of capacity constraints and parent component relationships. Consequently, decisions regarding lot sizing can be independently made for each item. This section delves into the formulations and solution procedures for two significant incapacitated lot sizing models, namely the Economic Order Quantity (EOQ) model (subsection 3.1) and the Wagner-Whitin model (subsection 3.2). While the EOQ and Wagner-Whitin algorithms may not be considered state-of-the-art in modern operations research, they persist as valuable benchmarks in the field [6]. Their enduring relevance lies in their widespread usage as reference points, providing a basis for comparison and evaluation when assessing the efficacy of more contemporary and sophisticated lot sizing models.

3.1. Economic Order Quantity

The EOQ model was initially introduced in 1913 by Harris with the statement: ”How Many Parts to Make at Once“ [7]. It can be seen as one of the earliest applications of mathematical modelling to scientific management. Since then, it has been a remarkable benchmark in the literature and is still widely used [8]. The EOQ model, formulated by Harris, represents a seminal contribution to inventory management and remains a fundamental concept in operations research and supply chain management [7, 8]. The EOQ model addresses the critical question of determining the optimal order quantity for minimizing total inventory costs, considering both holding costs and ordering costs.

In general, some assumptions have to be clarified for the EOQ model from Harris [7]. The notation is based on the work by Hopp and Spearman [9, p. 51]. The first assumption is that customer demand for the item is assumed to be known and constant at a rate **D**. Secondly, it should be assumed that for the EOQ model the leadtime is equal to zero, which means that delivery or manufacturing is instantaneous. With this assumption, there is no need to consider when an order should ideally be placed. The answer to this consideration describes the statement that **Q** is ordered every time the stock falls to zero. This can be summarised to the extent that it should be ordered Q when inventory equals a leadtime's supply [10]. In addition, three assumptions must be made regarding costs. The cost of the units themselves, denoted **c** is assumed to be fixed, regardless of the number of units ordered or manufactured. That means that there are no discounts for big purchases or other reductions. Finally, there are fixed manufacturing setup costs, denoted **A** and determined costs for holding items in the inventory, denoted **h**. The setup costs do not change regardless of the amount of products to be produced. Holding item costs h are defined as the product of the annual interest rate (**i**) and the capital invested in inventory (c), denoted by the equation $h = ic$, when the holding cost primarily comprises interest on the funds associated with inventory.

In the context of the EOQ model, the objective is to determine the order quantity (Q), which minimizes the average cost or time associated with inventory management arising from the act of placing orders for quantity Q whenever the inventory depletes to zero. In the following, the average cost arising from the order quantity Q is represented as **AC(Q)**. The decision variable of the entire EOQ model is therefore the unknown order quantity (Q). Assuming constant and deterministic demand, ordering Q units each time the inventory reaches zero yields an average inventory level of $Q/2$, as depicted in Figure 1. The annual holding costs can be described as

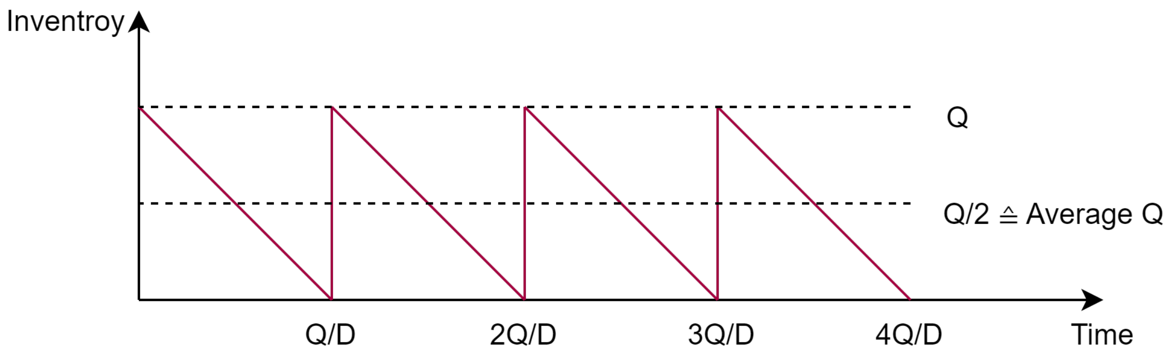


Figure 1: Inventory versus time in the Economic Order Quantity model [9].

$hQ/2$, whereby the unit holding costs can be calculated by $hQ/2D$. The setup costs has been already defined before to A, from which the unit setup costs can be calculated with A/Q. If the annual holding costs are now added to the unit setup costs and the production costs already defined above, the following cost function is obtained:

$$Y(Q) = \frac{hQ}{2D} + \frac{A}{Q} + c \tag{1}$$

Now that the cost function has been defined, the next goal is to minimise it. Thus, determining Q involves balancing the average ordering cost against the average inventory-holding cost. Observe in Figure 2 that the minimum value of AC(Q) happens where the graphs of average annual ordering costs and average annual inventory-holding intersect, specifically at $A * (D/Q) = h * (Q/2)$ [10].

$$A \times \frac{D}{Q*} = h \times \frac{Q*}{2} \tag{2}$$

Consequently, we can identify the optimal Q, represented as $Q*$, by solving Equation 2 for $Q*$. Solving this formula gives the following solution:

$$Q* = \sqrt{\frac{2 \times A \times D}{h}} \tag{3}$$

Equation 3 represent the final EOQ Square Root Formula, which is also referred to as the economic lot size [9, p. 53].

The EOQ model provides valuable insights into inventory management strategies, aiding busi-

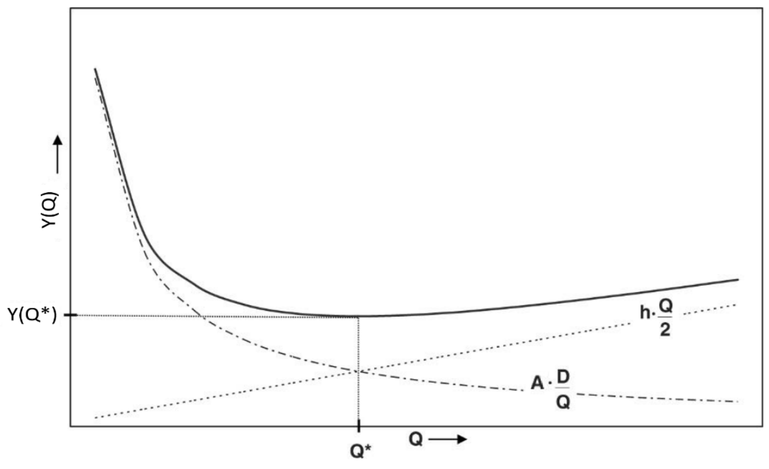


Figure 2: Costs in the Economic Order Quantity model. Based on [10, p. 141].

nesses in achieving cost-efficiency, improving cash flow, and enhancing overall operational performance. Over the years, the EOQ model has evolved and adapted to various business environments, remaining a cornerstone in the design and optimization of inventory systems across diverse industries.

3.2. Wagner-Whitin

The Wagner-Whitin model represents a further development in inventory management for moving beyond the static assumptions of earlier models like EOQ in subsection 3.1 [11]. Developed by Wagner and Whitin in 1958, this model is a dynamic version of the economic lot size model, addressing variable demand over a finite planning horizon. This dynamic model addresses the complexities of varying demand over a finite planning horizon, which is a critical aspect in inventory management practices [9]. Unlike the EOQ model, which assumes a constant demand rate, the Wagner-Whitin model divides the planning period into several discrete periods, acknowledging that demand can fluctuate significantly over time.

In mathematical terms, the Wagner-Whitin model can be viewed as a series of decisions for each period, where the decision to produce or is based on the trade-off between setup costs and holding costs. These decisions are made in the context of varying demand forecasts for each period, emphasizing the need for flexibility in inventory management. An important insight is that if production occurs in period t (including setup costs) to meet the demand of period $t+1$, then it would not be cost-effective to also produce in period $t+1$, as this would entail an additional setup cost. It has to be cheaper to produce the products of period t and $t+1$ in the first period. It is not economical to schedule production of each product in every individual period.

A unique feature of this model is its expression as the shortest path problem. This perspective enables efficient computational strategies to find optimal solutions, making the model not only theoretically grounded but also practically applicable in various inventory management scenarios. The Wagner-Whitin model deals with the problem of determining production lot sizes when demand is deterministic but time-varying and all other assumptions of the EOQ model are valid. Furthermore, the Wagner-Whitin model's dynamic approach to lot sizing has been influential in both academic literature on production control and the practical development of Materials Requirement Planning (MRP) systems [9].

Calculation

In order to understand the calculation of the Wagner-Whitin model, the following notations must first be defined [11]:

- $f_t(I)$ - the minimized cost function of a period t with a given inventory I in the beginning.
- i_{t-1} - the inventory cost per unit, if the product is stored until period t.
- x_t - manufactured amount in period t.
- s_t - setup costs for manufacturing in period t.
- d_t - the demand in period t.
- $\delta(x_t)$ - is an indicator function that is used to determine whether in a given period t an order is placed or not:
  - $\delta(x_t) = 1$, if $x_t > 0$, which means that there is production in period t, and therefore setup costs are incurred.
  - $\delta(x_t) = 0$, if $x_t = 0$, which means that there is no production in the period t, and hence no setup costs are incurred.

The Wagner-Whitin algorithm proceeds to calculate the optimal production schedule across the planning horizon. It does this by evaluating each period on its own, determining whether it is more cost-effective to produce in the current period or in a future period. Followed is Equation 4 which is needed to calculate the optimal lot size:

$$f_t(I) = \min[i_{t-1}I + \delta(x_t)s_t + f_{t+1}(I + x_t - d_t)] \tag{4}$$

Flow of the Wagner-Whitin Algorithm

The Wagner-Whitin algorithm advances sequentially from the first period to the final period, denoted as period T [9]. As the model progresses through each period, the algorithm evaluates the cost-effectiveness of producing in the current period versus a future period, accounting for the trade-offs between setup costs and holding costs. An optimal approach is characterized by each value of $\delta(x_t)$ precisely matching the sum of a series of future demands d_t. Followed, the possible solutions for the algorithm will be listed [9]:

$$\begin{array}{lllll}
\delta(x_1) = d_1 & \text{or} & \delta(x_1) = d_1 + d_2 & \text{or} & \delta(x_1) = d_1 + d_2 + ... + d_T \\
\delta(x_2) = 0 & \text{or} & \delta(x_2) = d_2 + d_3 & \text{or} & \delta(x_1) = d_1 + d_2 + ... + d_T \\
\vdots & & & & \\
\delta(x_T) = 0 & \text{or} & >\delta(x_2) = d_T & &
\end{array}$$

An exact requirement's management policy is fully specified by indicating the time periods in which the order should be placed. The number of exact requirements policies is much smaller than the total number of realizable policies. The sequential process is in Figure 3 displayed. The

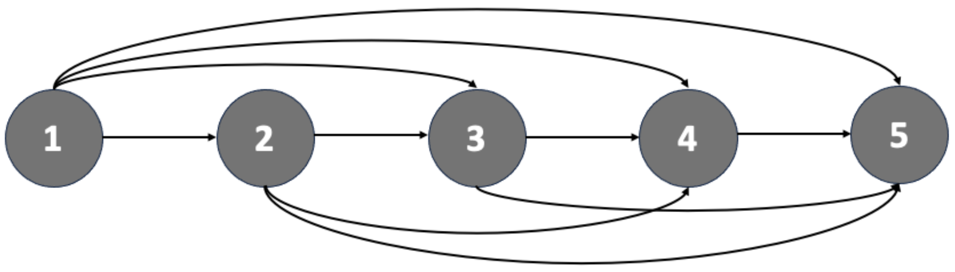


Figure 3: Combination representation for Wagner-Whitin. Based on [12, p. 486].

Wagner-Whitin algorithm conducts a forward calculation for each period t. In every period, it evaluates the potential production options, starting from the last production period up to the current period t. For each period, the algorithm assesses setup costs and holding costs.

If the most favourable option within the examined scope is to adhere to the existing production plan and maintain the last production date, this option is selected. Conversely, if initiating production in a more recent period proves to be more cost-effective, the management will opt for this alternative. When the production plan is updated, the algorithm, in its subsequent phase, will consider only two potential production periods.

This decision-making process is grounded in the principle of cost minimization, aiming to optimize the total costs across the entire planning horizon.

3.3. Theoretical Models Prospection

The EOQ model and the Wagner-Whitin algorithm have long been stalwarts in the realm of deterministic models for lot sizing, providing a solid foundation for research studies in inventory management [6]. Glock, Grosse, and Ries continue that various extensions of Harris lot size model have been developed over the years, including models that incorporate multi-stage inventory systems, incentives, and productivity issues [6, 7]. Despite the evolving landscape of supply chain dynamics and the advent of more sophisticated models, these classic frameworks continue to captivate the attention of researchers for several compelling reasons.

Its simplicity and intuitive appeal make it an attractive starting point for scholars exploring lot sizing problems. Researchers often use the EOQ model as a benchmark or baseline against which they can compare the performance of more intricate models, helping to assess the practical significance and real-world applicability of newer methodologies [6].

On the other hand, the Wagner-Whitin algorithm extends the scope of lot sizing models by considering production capacity constraints and dynamic demand patterns. This algorithm, building upon the EOQ principles, introduces a finite production rate and allows for backlogging, thus capturing more realistic scenarios. Its enduring relevance lies in its ability to address complexities beyond the scope of the original EOQ model, making it a valuable tool for researchers delving into more nuanced lot sizing problems.

Various extensions of Harris lot size model have been developed over the years, including models that incorporate multi-stage inventory systems, incentives, and productivity issues [6]. In summary, the enduring appeal of the EOQ model and the Wagner-Whitin algorithm in research studies on deterministic lot sizing models can be attributed to their simplicity, practicality, and relevance in addressing real-world inventory challenges. These models serve as essential building blocks, providing a solid starting point for researchers to explore and develop more sophisticated approaches in the ever-evolving field of supply chain management.

3.4. Current Innovations in Lot Sizing Techniques

Lot sizing approaches nowadays are focusing on a more advanced problem which isn't capable with EOQ or Wagner-Whitin. This shift towards advanced models is driven by the increasing complexity of manufacturing environments, where hybrid flow shop systems are becoming more prevalent. The publication "Reformulation, linearization, and a hybrid iterated local search algorithm for economic lot-sizing and sequencing in hybrid flow shop problems" is researching a new model of solving the lot sizing problem [13]. Their study introduces a novel Mixed-Integer Nonlinear Programming (MINLP) model, addressing the unique challenges posed by these hybrid systems. This is a production system featuring multiple parallel machines at one or more stages of the production process, enhancing flexibility and efficiency in handling diverse product types and processing requirements. Their approach integrates economic lot sizing with production sequencing, optimizing both inventory levels and production flow. The MINLP model improves an existing model, and presents a more manageable technique. The research demonstrates significant improvements in solvability and optimality gaps over existing models and algorithms, particularly for large-size instances, highlighting a state-of-the-art solution in lot sizing models.

4. Project Development Framework

The goal of this project is to develop an application that provides a web-based frontend for calculating the EOQ and the Wagner-Whitin algorithm. This frontend will transfer data to a backend, which handles the actual computation.

Various frameworks are integrated for both the front- and the backend. The term 'Framework' refers to a semicomplete application that offers a reusable and shared common structure for applications. Developers integrate the framework into their applications, customizing and extending it to fulfil their specific requirements. In contrast to toolkits, frameworks extend their functionality beyond the provision of utility classes by providing a cohesive structure that integrates various components seamlessly. The developer's code is appropriately invoked by the framework, allowing developers to concentrate less on the quality of the design and more on the implementation of domain-specific functions [14, p. 4].

4.1. Methodical Toolbox

To implement such a program with front- and backend, a couple of different software applications is needed. Beside, a decision for specific programming languages fitting Integrated Development Environment (IDE) has to be chosen. When the decision for the programming languages is done, specific middleware which can provide an interface between front- and backend has to be chosen. The selected software is described below.

4.2. GitHub

GitHub[2], as a cornerstone of the project development framework, underscores its pivotal role in the orchestration of collaborative software development. This platform is known for its robust version control and collaborative features. Which provides the required functionalities for this project. GitHub was used as a key tool for sharing, reviewing, and merging code in this project. It helped keep the project organized by allowing the team to work on different parts of the application at the same time. For instance, while some worked on the front end, others could focus on the back end or the calculation algorithms, without interfering with each other's work. This was managed through the use of branches and pull requests, ensuring that changes can be retraced after they were added to the main project.

4.3. Java

Java[3], a versatile and widely-used programming language, is renowned for its platform independence, robustness, and scalability. Its object-oriented nature and extensive standard libraries contribute to the language's popularity for building diverse applications, ranging from web and mobile to enterprise systems. Implementing EOQ and Wagner-Whitin models in Java make sense due to the language's platform independence, robust numerical computation capabilities, and extensive libraries, providing a scalable and efficient solution for inventory management that can be easily maintained and integrated across diverse systems.

As IDE the decision was made for IntelliJ IDEA[4], which describes themselves as leading Java IDE. IntelliJ IDEA is a comprehensive suite of tools and features tailored for Java programming, streamlining the development process by offering advanced code editing, syntax highlighting, and code completion, thereby enhancing coding accuracy and efficiency. Additionally, it offers robust debugging capabilities, allowing the developers to identify and rectify errors swiftly. IntelliJ IDEA provide real-time error checking, breakpoints, and step-through debugging, facilitating a more systematic and time-effective approach to identifying and fixing bugs in Java code.

4.4. JUnit Tests

Software verification is an integral aspect of the development process, wherein each code iteration undergoes rigorous testing procedures. Initiated by the programmer's acceptance test, the development cycle involves coding, compiling, and subsequent execution, constituting a comprehensive

[2]https://github.com/ accessed on 23.01.2024
[3]https://www.java.com/en/ accessed on 27.11.2023
[4]https://www.jetbrains.com/idea/ accessed on 27.11.2023

evaluation. Daily iterations encompass coding activities, compilation processes, execution, and meticulous testing, often involving interactions such as button-click simulations or result analysis to validate expected outcomes.

JUnit[5] is an open source software hosted on GitHub, with an Eclipse Public License [14, p. 5]. Originally, the term “unit test” referred to a test that focused on checking the functionality of a single component, such as a class or a method. Mainly, it confirms that the method accepts the expected range of input and that the method returns the expected value for each input. If the value x is given, will it return value y? If the value z is given instead, will it throw the proper exception?

In the project, JUnit tests are used to check various functions over time, particularly in the backend. The main purpose is to automatically verify whether newly introduced changes are causing issues in other parts of the project. In addition to the main folder, the src folder also contains the test folder. The src folder can be found at the following path:

```
\DeterministicModelsLotSizing\javadeterministicmodelslotsizing\src\
```

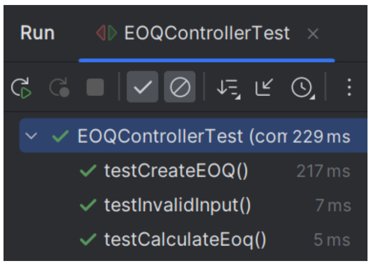
All classes required for the implemented tests can be found in this folder. In particular, two classes were implemented for testing. The classes were divided into the EOQ and Wagner-Whitin distinctions. The two classes each check all functions of the two classes. In addition to the calculation, this also includes the corresponding controller class, which provides the communication between the front and back end. The EOQ test classes are explained in more detail below. Similar tests were also written for the calculation and transmission of Wagner-Whitin.

The first given Java class is a JUnit test class named, `EOQControllerTest` designed to test the functionality of an EOQ controller within a Spring Boot application. The class is annotated with `@SpringBootTest` and `@AutoConfigureMockMvc`, indicating its status as a Spring Boot test with autoconfigured MockMvc. The necessary imports are included for JUnit testing, Spring Boot testing, Mockito, and other required classes. The class contains three test methods: `testCreateEOQ`, `testInvalidInput`, and `testCalculateEoq`. The `EOQControllerTest` class includes fields such as `EOQController eoqController` (a mock bean for the EOQController), `MockMvc mvc` (a MockMvc instance for Hypertext Transfer Protocol (HTTP) request/response testing), and `EOQ eoq` (an instance of the EOQ class). The setup method, annotated with `@BeforeEach`, is executed before each test. It initializes a new EOQ instance. The `testCreateEOQ` method tests the calculate method of `EOQController` when a valid EOQ object is provided. It calls the `eoqMethod` of the EOQ instance with specific parameters, uses Mockito's given to mock the response of `eoqController.calculate()`, and expects a JavaScript Object Notation (JSON) response with the value 77.0. The `testInvalidInput` method tests the behaviour of `EOQController` when an invalid input is provided. It uses Mockito to mock the response of `eoqController.calculate()` and expects a `BAD_REQUEST` (400) status. The `testCalculateEoq` method tests the `eoqMethod` of the EOQ class with different input parameters. It utilizes JUnit's `assertEquals` to verify that the calculated values match the expected values. In summary, this test class aims to ensure the proper functioning of the EOQ controller's calculate method for both valid and invalid inputs. Additionally, it includes tests for

[5]https://junit.org/junit5/ accessed on 25.01.2024

the underlying calculation logic in the EOQ class.

An excerpt from the test class for the EOQ is shown below as an example. If the `EOQControllerTest.java` class is executed, a Spring Boot application is started automatically, which then performs three tests. The three tests are divided into the three functions `testCreateEOQ()`, `testInvalidInput()` and `testCalculateEOQ()`, which are described above. The feedback of the successful tests can be seen in Figure 4.



```
✓ EOQControllerTest (con 229 ms
    ✓ testCreateEOQ()      217 ms
    ✓ testInvalidInput()     7 ms
    ✓ testCalculateEoq()     5 ms
```

Figure 4: Successfully completed JUnit Tests.

4.5. Apache Maven

Apache Maven[6] is a software project management tool, which relieves the software development process and also unifies the development procedure [15]. The foundation about Maven is the Project Object Model. This `pom.xml` is a file inside the dedicated project. Apache Maven will be utilized for its superior project management and build automation capabilities. By simplifying the build process and ensuring consistent project structure, it will play a crucial role in maintaining the integrity and efficiency of the development workflow. Maven is normally used in the command line, but IntelliJ IDEA supports a fully-functional integration inside the project[7].

4.6. Spring Initializr

Spring Initializr[8] is a convenient, web-based tool designed to streamline the process of setting up a new Spring Boot project [15]. A Spring Boot project is essentially an application that simplifies the development process by providing a framework for creating stand-alone, production-grade applications quickly and with minimal configuration. The Initializr offers a user-friendly interface where developers can quickly generate and download a basic project structure, tailored to their specific needs. By selecting desired dependencies, Java version (see subsection 4.3), and build tools like Maven (see subsection 4.5), developers can create a ready-to-use Spring Boot project with minimal setup. This initial setup includes all the necessary configurations and dependencies, allowing developers to immediately start working on the business logic of their application, rather than spending time on initial project setup and configuration. Spring Initializr acts as an essential starting point for building robust, efficient Spring Boot applications, significantly reducing initial development time and complexity.

[6]`https://maven.apache.org/` accessed on 26.01.2024
[7]`https://www.jetbrains.com/help/idea/maven-support.html` accessed on 27.11.2023
[8]`https://start.spring.io/` accessed on 29.01.2024

4.7. REpresentational State Transfer

REpresentational State Transfer (REST) is an architectural style in web services development and was conceptualized by Roy Fielding [16]. It operates under a set of principles emphasizing simplicity and scalability in networked applications, closely intertwining with the concept of an Application Programming Interface (API) [15]. An API acts as a conduit for communication between different software components, often dictating the rules and methods for data exchange. This allows the application to efficiently handle requests and transmit data in a format that is both developer-friendly and optimized for performance. In RESTs context, the API is the medium through which clients and servers interact, typically via HTTP protocols. REST is characterized by its stateless nature, meaning the server does not retain client state information, thus ensuring that each interaction is independent and self-contained. This principle of REST contributes to a distinct separation between client and server, enhancing both the independence and scalability of the system, which aligns seamlessly with the fundamental role of an API in facilitating flexible and efficient client-server communication.

4.8. React.js

Now that the backend and the appropriate interface, which is described in subsection 4.7, have been defined, the technical conditions for the frontend are described below. The calculations from the backend should be displayed to the end user using a website. Input and output should therefore be possible for the user on this website. The plan is to develop a website based on Hypertext Markup Language (HTML)[9]. In order to display content more dynamically, React.js in particular is to be used, which in turn is based on JavaScript, an HTML extension.

In the realm of frontend development, choosing React.js[10] emerges as a strategic decision driven by its distinctive advantages. React's declarative syntax empowers developers to concisely articulate User Interface (UI) components, facilitating a more straightforward and intuitive development process. The component-based architecture promotes code modularity, enabling the creation of reusable and maintainable building blocks for complex user interfaces. React's Virtual Document Object Model implementation optimizes performance by minimizing unnecessary re-rendering, ensuring swift and efficient updates. With an extensive and collaborative community, React.js offers a wealth of resources, support, and pre-built components, expediting development and troubleshooting. Backed by Facebook, React.js not only exemplifies reliability but also aligns with industry best practices, making it a compelling choice for developers aiming to streamline and elevate their frontend projects. The architecture from subsection 4.3, subsection 4.7 and the acutal subsection can be seen in Figure 5.

4.9. Postman

Postman[11] is an application which offers multiple tools to test REST-APIs [15]. Notably, it simplifies the process of API testing by allowing users to create and save collections of requests, which can be reused and shared among team members. It offers a user-friendly interface for

[9]`https://html.spec.whatwg.org/` accessed on 28.11.2023
[10]`https://react.dev/` accessed on 29.11.2023
[11]`https://www.postman.com/` accessed on 02.02.2024

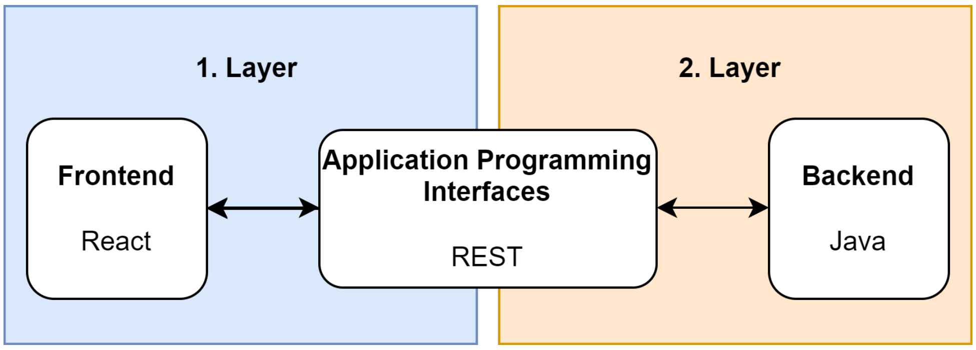


Figure 5: Software architecture.

sending requests to web servers and viewing responses [16]. This feature is particularly useful for demonstrating API functionality to stakeholders who may not be technically adept. Additionally, Postman supports various types of HTTP requests, authentication methods, and has an extensive documentation. Its built-in tools for testing and validating API responses ensure that APIs meet their intended specifications. Furthermore, Postman facilitates environment management, enabling users to develop and test their APIs in different configurations, thereby enhancing the robustness and reliability of API services. The application also integrates with continuous integration and continuous deployment pipelines, automating the API testing process in different stages of software development.

4.10. Swagger

Swagger[12] is a pivotal tool in software development, primarily used for designing, documenting, and testing REST-APIs [17]. It serves as a bridge between various teams involved in software development, including developers, testers, and business analysts. This application enhances understanding among stakeholders without requiring deep dives into source code. Moreover, its intuitive interface simplifies the process of creating and managing API endpoints, making it accessible even to those with limited technical expertise. Additionally, Swagger UI offers an interactive platform for executing API requests and visualizing responses, aiding in debugging and demonstration. The tool's capacity to facilitate the integration of APIs into various applications through its support for multiple programming languages significantly streamlines the development process. Additionally, Swagger's broad compatibility with a range of platforms and programming languages amplifies its adaptability in a variety of development contexts.

Despite the versatility and utility of Swagger in software development, the team opted not to implement it for API testing purposes. This decision stemmed from the comprehensive capabilities of Postman (see subsection 4.9), which proved sufficient for the testing needs. Given Postman's robust features for testing APIs, fulfilled the requirements without the need for additional tools such as Swagger. Additionally, leveraging Postman allowed for seamless collaboration among team members and provided a user-friendly interface for demonstrating API functionality to stakeholders. Consequently, the reliance on Postman obviated the necessity for integrating

[12]https://swagger.io/ accessed on 01.02.2024

Swagger into the development workflow.

4.11. Expected Outcome

The expected outcome of this project is the development of an advanced web-based application tailored for businesses to effectively calculate the EOQ and implement the Wagner-Whitin algorithm. The application let the users decide which of the calculation approaches they want to use. Based on this selection, the UI changes. This flexibility ensures that businesses of various sizes and with different operational complexities can benefit from the tool. The dynamic nature of the UI enhances the user engagement and provides a tailored experience for different calculation methods. The data will be transferred with REST-API from the frontend to the backend and vice versa.

This tool is designed to offer an easy-to-use application for inventory management and lot sizing. With its user-friendly interface, the application allows users to input data effortlessly and receive accurate, optimized calculations. Its development is expected to bridge the gap between complex inventory management theories and practical, real-world application. The integration of robust frontend and backend technologies ensures a seamless user experience and reliable results. This application will be beneficial in a business context, aiding companies in making strategic decisions about inventory management, leading to significant cost savings, improved resource allocation and enhanced operational efficiency.

5. Backend Implementation

Followed, the framework of the backend will be discussed, which includes an overview of the main structure and the methodology used to calculate the two deterministic models for lot sizing.

5.1. Implementing the REpresentational State Transfer Application Programming Interface

As the backend is based on Spring Boot and RESTful API, the idea is to build an environment, which is potentially capable to scale the size of the project and still give the possibility of a quick overview. The Controllers have the role of a middleware between the frontend and the backend. They manage the flow of data and contribute to a more organized backend by centralizing communication outside the core portion of the backend.

In Spring Boot, controllers are designed as Java classes annotated with the `@RestController` annotation. This shows they handle HTTP requests. The controller contains multiple methods, each executing a dedicated task focused on handling crucial parts of communication within the class. This structure streamlines request processing, making the backend more organized. Additionally, the use of specific annotations for each method ensures that requests are accurately routed to their corresponding handlers. An essential element of every method in a controller is the annotation that specifies the web link the method is accessible through. For large input and output values, setting the format is also recommended to ensure the application adheres to it.

Every method must indicate the type of methodology, which can be defined using the annotations: `@GetMapping`, `@PostMapping`, `@PutMapping` and `@DeleteMapping`. These annotations embody the "Create, Read, Update, Delete" ideology, offering a structured approach to handling data operations. This methodology not only simplifies development but also enhances the maintainability of the application by clearly defining how resources are managed.

The controller for transferring the values of the EOQ is shown in subsection A.1, and the controller for the Wagner-Whitin is shown in subsection A.2.

5.2. Calculation of the Economic Order Quantity

The implementation of the calculation of EOQ in the backend in Java turned out to be a very simple method. The theory behind the calculation can be viewed in the Equation 3 in subsection 3.1. To begin with, a EOQ-class is created, which receives the required input values as variables. This class was then supplemented with the `eoqMethod(double setDemand, double setA, double setH)`, which performs the calculation. After assigning the transfer values of the method, the entire calculation is performed in one line and finally rounded and returned by the method. For a detailed review of the calculation process of the EOQ model, see Appendix A.3.

5.3. Calculation of the Wagner-Whitin

The calculation process of the Wagner-Whitin model, as outlined in class `WagnerWhitinAlgorithm`, involves a series of steps aimed at solving the Wagner-Whitin Algorithm. Followed will be a step-by-step explanation about the calculation itself.

1. **Initialization of Cost Arrays:** The method begins by initializing two key arrays: `totalCost` and `orderSchedule`. `totalCost` is filled with the maximum value an integer can hold. This step sets the initial costs high so that any actual cost found later will be lower, this is helping to identify the lowest cost more easily. Similarly, `orderSchedule` is filled with -1 to indicate that no production schedule has yet been determined. This setup prepares the algorithm to track the optimal costs and corresponding production schedules.

2. **Filling the First Row of the Cost Matrix:** Next, the `costMatrix` will be initialized which is a Two-Dimensional Array. The first row of the `costMatrix` is filled with the costs of starting production in the first week and carrying inventory for all following weeks. Inside this `CostMatrix` the ordering cost summed up for the whole time with the holding costs will be stored.

3. **Dynamic Cost Calculation:** The algorithm is iterating through each week as a potential start week for production, the costs for different production schedules will be calculated each. This considers each week as a potential start point, calculates holding costs from that week to the current week in the iteration, and compares these calculated costs to the existing costs in the `totalCost` array to determine if a more cost-effective solution exists.

4. **Updating Cost and Schedule Arrays:** If the calculated cost for starting production in a specific week is lower than the current total cost for that week, the algorithm updates

the `totalCost` and `orderSchedule` arrays. This ensures the finding of the most cost-efficient production schedule.

5. **Adjusting the Cost Matrix:** The algorithm adjusts the cost matrix to highlight the optimal production schedules determined. This involves setting irrelevant values in the `costmatrix` to the value zero to make the result easier to interpret. This behaviour is also given in the book *Factory physics* by Hopp and Spearman [9].

6. **Printing the Results:** The method concludes by printing the optimal production periods, the adjusted cost matrix, and the minimum total costs to the integrated console. This final step provides the possibility to check the calculation even without an UI.

Each step in this process iteratively refines the production schedule to minimize total costs, including both ordering and holding costs. The systematic consideration of all possible production start weeks ensures the final solution is optimal for the entire planning horizon. For a detailed explanation of the calculation process of the Wagner-Whitin model, see Appendix A.4.

Additionally, two Java classes were created for the Request and the Response. These mainly consist of a Constructor and some Getters and Setters. The background behind the additional classes were to make the interaction with the frontend through the controller more generalizable.

6. Frontend Implementation

The frontend development phase was a pivotal part of this project, focusing on creating a user-friendly interface that allows seamless interaction with the backend's complex algorithms. Within the following text, the frontend of the application will be explained.

6.1. Bootstrap Template for the Frontend

In the development of the project, the decision was made to employ a pre-designed template for the frontend interface. React is known for its flexibility and wide range of options for template integration, facilitated the adoption of Bootstrap[13]. Bootstrap was selected for its comprehensive framework that supports responsive design and accelerates the development of user interfaces.

The template chosen for this purpose was the “Light Bootstrap Dashboard React” by Creative Tim[14]. This template was selected for its extensive collection of components and its customizability to meet specific project needs. The utilization of this template significantly reduced development time, allowing for a focus on enhancing functionality rather than constructing a design from scratch. It provided a robust starting point with an intuitive and modern UI, thus improving the end-user experience. The implementation of the Bootstrap-based template not only helped to create an aesthetically pleasing design, but also a layout that adapts to different screen sizes and devices, ensuring accessibility and usability on different platforms.

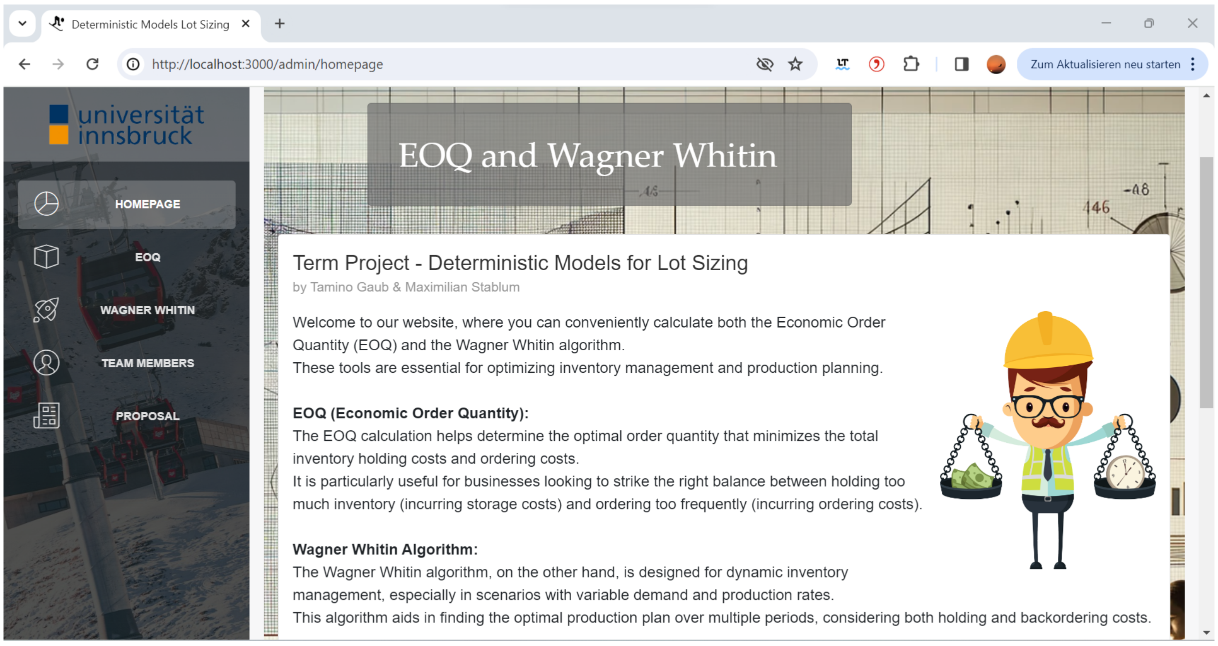
[13]https://getbootstrap.com/ accessed on 08.02.2024
[14]https://www.creative-tim.com/product/light-bootstrap-dashboard-react accessed on 05.02.2024

For this project, the main design of the template was used and the different views of the components were adjusted. Additionally, elements like the Footer and the Sidebar got adjusted to fit the criteria. This contains, editing the background of the Sidebar to a picture of the Axamer Lizum Gondola to fit better to the project at the Leopold-Franzens University Innsbruck. Additionally, the footer got adjusted with the respected Uniform Resource Locator (URL) of the GitHub Repository of the project and an option to contact the contributors of this project.

6.2. Adjustment of the Components

Followed, the different components of the project and their basic frontend functionality will be explained. The components which were already provided by the template got deleted completely.

- **Homepage:** The Homepage of the website consists of general information about the project and a small introduction about the deterministic models which are aimed to calculate. Also, the difference between the two models is explained to make sure, everybody is informed when to use which model. The final Homepage can be seen in Figure 6.


Deterministic Models Lot Sizing
http://localhost:3000/admin/homepage
Zum Aktualisieren neu starten
universität innsbruck
HOMEPAGE
EOQ
WAGNER WHITIN
TEAM MEMBERS
PROPOSAL
EOQ and Wagner Whitin

Term Project - Deterministic Models for Lot Sizing

by Tamino Gaub & Maximilian Stablum

Welcome to our website, where you can conveniently calculate both the Economic Order Quantity (EOQ) and the Wagner Whitin algorithm.
These tools are essential for optimizing inventory management and production planning.

EOQ (Economic Order Quantity):
The EOQ calculation helps determine the optimal order quantity that minimizes the total inventory holding costs and ordering costs.
It is particularly useful for businesses looking to strike the right balance between holding too much inventory (incurring storage costs) and ordering too frequently (incurring ordering costs).

Wagner Whitin Algorithm:
The Wagner Whitin algorithm, on the other hand, is designed for dynamic inventory management, especially in scenarios with variable demand and production rates.
This algorithm aids in finding the optimal production plan over multiple periods, considering both holding and backordering costs.

Figure 6: The final Homepage.

- **EOQ:** The EOQ-Component gives the following input criteria for the user:
  - Demand
  - Holding Costs
  - Setup Costs

  If the values are entered on the website (Frontend), the website first checks them. Negative and zero values are not permitted. If the check is positive, the data is sent to the backend, calculated there and then the calculated EOQ is returned and displayed on the website. It is also possible to download an Excel template. This offers the option of entering the demand spread over several periods. When the filled template is uploaded to the website, the average demand is calculated and the steps described above are then carried out. The template already checks for an authorised entry during input.

- **Wagner-Whitin:** The Wagner-Whitin Component has similar input values, like the EOQ, but the demand is split in periods. So the user can insert variable demands and also a dynamic amount of periods. The amount of periods can be adjusted flexible. The same possibility as for the EOQ-Component for the Excel upload is also provided for the Wagner-Whitin.

  The Results are printed below the input fields in the form of a table to make it easy to interpret. Additionally, the periods of production and maximum costs are displayed above the table.

- **Team:** The Team-Component is to present the contributing team, it consists of the Name, E-Mail, University and LinkedIn information. A picture of the two developers is also attached to the website. The team also provided their favourite personal song, to give the user a possibility to distract a few seconds after calculating the complex models.

- **Proposal:** The Proposal gives the user the possibility to read through the theory of the deterministic models. The UI provides an embedded PDF-File of the proposal, which was the theoretical basis for this project.

- **Final Report:** The Final Report component offers users the full documentation of this project, incorporating both the theoretical groundwork laid out in the proposal and the practical developments achieved.

6.3. Structure of Service classes

In the backend, the service class in the frontend acts as a counterpart to the controller, managing the exchange of data with it. This class holds the base URL for the controller. If the service class has multiple methods, it allows for the end of the URL to be changed dynamically. Service classes play a crucial role in organizing and managing data flow between the frontend and backend, encapsulating business logic and data access. They act as intermediaries, ensuring that data is correctly prepared for the frontend and efficiently processed by the backend. This structure enhances modularity and maintainability, as changes to the base URL require updates in only one place, rather than in each method. In scenarios where the base URL changes, there's no need to update the link for every method individually, simplifying maintenance and development.

7. Challenges Encountered in Application Development

During the development of the project, several problems were faced, which will be documented in the following part. The problems were of various types.

7.1. Challenges in Connecting Front- and Backend with REpresentational State Transfer Application Programming Interface

The goal was to be able to send data from the Java backend to the React frontend and vice versa. Within this process of combining these two components with RESTful API three issues occurred.

1. Misunderstandings in the Application Programming Interface Specification

Initially, a successful connection between the frontend and backend could not be established, as the API specifications were unclear. This lack of clarity led to misunderstandings about the required endpoints and the proper structure of requests and responses. Consequently, a series of errors in HTTP requests occurred. To address these issues, Postman (see subsection 4.9) was used to fetch and validate the output of the HTTP requests, which was instrumental in identifying and correcting the problems, thereby improving the communication between the frontend and backend.

The decisive turn came with the introduction of Axios[15] for HTTP requests, enabling efficient and error-free communication between the frontend and backend. The data formats were standardized, and the JSON data was carefully structured to ensure consistency and compatibility. With these changes, a seamless and powerful connection between the React frontend and the Java backend was finally established.

2. Issues with Cross-Origin Resource Sharing

When trying to send requests from the React frontend to the Java backend, challenges related to Cross-Origin Resource Sharing (CORS) were encountered. CORS is a security feature in web browsers that prevents web applications from making requests to a domain other than that of the web application. This security measure is important to prevent unwanted cross-site scripting and data breaches. In this project, as the frontend is running under the domain: https://localhost:3000 and the backend is running under https://localhost:8080, the CORS policy is blocking their interaction by default. The solution for this issue was to add the following code snippet into the Backend inside the Spring Boot Application Class:

```
@Bean
public WebMvcConfigurer corsConfigurer() {
    return new WebMvcConfigurer() {
        @Override
        public void addCorsMappings(CorsRegistry registry) {
            // Apply CORS settings to all endpoints ("/**")
            registry.addMapping("/**").allowedOrigins("http://localhost:3000");
        }
    };
}
```

[15] https://axios-http.com/ accessed on 08.02.2024

3. Inconsistencies in Data Type Handling

When transferring JSON data from the Java backend to the React frontend, problems occur with the exact adjustment of the data format. This discrepancy led to difficulties in processing the data correctly once it reached its destination. Interestingly, the compiler did not indicate that getters were missing, which contributed to this challenge. The data, which was appropriately structured in the backend, was not transferred as intended in the React environment. To resolve this, specific getters were defined within the Java classes. Implementing these getters ensured that the correct data was sent to the frontend, and it was in a format that the React components could understand and process correctly, solving the data processing issues. The package which was not in a proper JSON format could be identified also using the Postman, so the point of the error could be enclosed to the backend.

7.2. GitHub Merge Conflict

During the development phase, a GitHub merge conflict emerged when attempting to merge the branches `Connection_RESTful_Wagner_Whitin` and `Tamino_EOQ`. This conflict was due to simultaneous modifications to the same files in both branches, resulting in discrepancies that GitHub's automatic merge process could not resolve. After the merge attempt, the project's functionality was significantly impacted, with only partial integrations from both branches. The objective then shifted towards reverting these changes, a task that proved unexpectedly complex. Identifying the specific commits that caused the disruption became a significant challenge, complicating the process of returning to a stable project state. It also emphasized the benefits of making more granular commits, which would allow for easier tracking and reversal of changes if conflicts arise. Ultimately, resolving the conflict required careful use of git commands to revert changes and manually address the discrepancies, enhancing the understanding of effective version control strategies.

8. Lessons Learned within this Project

During the development process of the deterministic models for lot sizing application, the project unfolded a range of insights into constructing an End-to-End application. Although the scope of the application was relatively modest, the basic principles established offer the potential to expand the project to a larger scale.

A primary takeaway was the essential role of theoretical foundation. Implementing deterministic models, especially the Wagner-Whitin algorithm, required a good understanding of their mathematical foundations. This need to link theoretical concepts with practical applications underlines the importance of software solutions based on theoretical scientific foundations. Moreover, the project illuminated the significance of teamwork and its contribution to the developmental stride. Ensuring effective collaboration and seamless communication was crucial, keeping both contributors in synchronisation with the current project's progress and challenges. To maintain consistent communication, team members discussed the project at regular intervals.

In sum, the project was a holistic learning journey, emphasizing the synergy between theoretical insights and practical execution, the necessity for both technological and procedural adaptability, the dynamics of effective teamwork, and the paramount importance of prioritizing user experience.

References

[1] G. De Souza Amaro, D. J. Fiorotto, and W. A. De Oliveira. "Impact analysis of flexibility on the integrated lot sizing and supplier selection problem". In: *TOP* 31.1 (July 2022), pp. 236–266. DOI: 10.1007/s11750-022-00636-2.

[2] P. Robinson, A. Narayanan, and F. Sahin. "Coordinated deterministic dynamic demand lot-sizing problem: A review of models and algorithms". In: *Omega* 37.1 (2009), pp. 3–15. ISSN: 0305-0483. DOI: https://doi.org/10.1016/j.omega.2006.11.004.

[3] E. Farhi et al. "A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem". In: *Science (New York, N.Y.)* 292.5516 (2001), pp. 472–475. ISSN: 0036-8075. DOI: 10.1126/science.1057726.

[4] A. Drexl and A. Kimms. "Lot sizing and scheduling — Survey and extensions". In: *European Journal of Operational Research* 99.2 (1997), pp. 221–235. ISSN: 03772217. DOI: 10.1016/S0377-2217(97)00030-1.

[5] M. Salomon. *Deterministic lotsizing models for production planning.* Vol. 355. Lecture notes in economics and mathematical systems. Berlin: Springer, 1991. VII, 158. ISBN: 978-3-540-53701-4.

[6] C. H. Glock, E. H. Grosse, and J. M. Ries. "The lot sizing problem: A tertiary study". In: *International Journal of Production Economics* 155 (2014), pp. 39–51. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2013.12.009.

[7] F. W. Harris. "How Many Parts to Make at Once". In: *Operations Research* 38.6 (1990). (reprint from Factory - The Magazine of Management 10 (1913) 135-136, 152), pp. 947–950. ISSN: 0030-364X. DOI: 10.1287/opre.38.6.947.

[8] D. Erlenkotter. "Ford Whitman Harris and the Economic Order Quantity Model". In: *Operations Research* 38.6 (1990), pp. 937–946. ISSN: 0030-364X. DOI: 10.1287/opre.38.6.937.

[9] W. J. Hopp and M. L. Spearman. *Factory physics.* 3. ed. Long Grove, Ill.: Waveland Press, 2011. XXV, 720. ISBN: 1-57766-739-5.

[10] L. B. Schwarz. "The Economic Order-Quantity (EOQ) Model". In: *Building Intuition.* Ed. by F. S. Hillier, D. Chhajed, and T. J. Lowe. Vol. 115. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2008, pp. 135–154. ISBN: 978-0-387-73698-3. DOI: 10.1007/978-0-387-73699-0_8.

[11] H. M. Wagner and T. Whitin. "Dynamic version of the economic lot size model". In: *Management Science* 5 (1958), pp. 89–96.

[12] S. Nahmias and T. L. Olsen. *Production and Operations Analysis.* 7th ed. Waveland Press, Inc., 2015, pp. 485–489. ISBN: 9781478623069.

[13] H. Zohali et al. "Reformulation, linearization, and a hybrid iterated local search algorithm for economic lot-sizing and sequencing in hybrid flow shop problems". In: *Computers & Operations Research* 104 (2019), pp. 127–138.

[14] C. Tudose. *JUnit in Action.* Third edition. 2020. ISBN: 9781617297045.

[15] J. Hinkula. *Full Stack Development with Spring Boot and React: Build modern and scalable web applications using the power of Java and React.* Vol. 3. May 2022. ISBN: 9781801818643.

[16] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés. “Online Testing of RESTful APIs: Promises and Challenges”. In: ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, pp. 408–420. ISBN: 9781450394130. DOI: 10.1145/3540250.3549144.

[17] H. Wu et al. “Combinatorial Testing of RESTful APIs”. In: New York, NY, USA: Association for Computing Machinery, 2022, pp. 426–437. DOI: 10.1145/3510003.3510151.

A. Appendix

A.1. Economic Order Quantity Controller

```java
    @PostMapping("eoq/calculation")
        public ResponseEntity<Double> calculate(@RequestBody EOQ request) {
            // Response generation based on the RequestBody
            System.out.println("getA: " + request.getaSetup());
            System.out.println("getD: " + request.getAverageDemand());
            System.out.println("getH: " + request.getH());

            // Check if any of the values is <= 0
            if (request.getaSetup() <= 0 || request.getAverageDemand() <= 0
                || request.getH() <= 0) {
                return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(
                    null);
            }

            // Calculate the EOQ with help of the algorithm
            EOQ eoq = new EOQ();
            double result = eoq.eoqMethod(
                    request.getAverageDemand(),
                    request.getaSetup(),
                    request.getH()
            );
            // Return the response to the console
            System.out.println("Rounded result for frontend: " + result);

            // Return the result with HTTP Status 200 (OK)
            return ResponseEntity.ok(result);
        }
```

A.2. Wagner-Whitin Controller

```java
@PostMapping(path = "/wagner_whitin/calculation", consumes = "application/json", produces = "application/json")
public ResponseEntity<WagnerWhitinResponse> calculate(@RequestBody WagnerWhitinRequest request) {
    try {
        // Response generation based on the RequestBody
        WagnerWhitinAlgorithm algorithm = new WagnerWhitinAlgorithm(
                request.getDemands(),
                request.getHoldingCostPerUnitPerPeriod(),
                request.getOrderCost()
        );
        // Calculate the Wagner-Whitin with help of the algorithm
        algorithm.calculate();
        // Map the values into the response format
        WagnerWhitinResponse response = new WagnerWhitinResponse(
                algorithm.getTotalCost(),
                algorithm.getOrderSchedule(),
                algorithm.getCostMatrix(),
                algorithm.getProductionPeriods()
        );
        // Return the response
        log.info("Calculation successful for request: {}", response);
        // Return the response with the status code 201
        ResponseEntity responseEntity = new ResponseEntity<>(response, HttpStatus.CREATED);
        return responseEntity;
    } catch (Exception e) {
        // Return the response with the status code 500
        log.error("Error during calculation: ", e);
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

A.3. Detailed Calculation of the Economic Order Quantitiy

```
public long eoqMethod(double setDemand, double setA, double setH){
    averageDemand = setDemand;
    aSetup = setA;
    h = setH;
    optimalQ = Math.sqrt((2* aSetup * averageDemand)/h);
    System.out.println("Economic order quantity EOQ is: " + optimalQ);
    return Math.round(optimalQ);
}
```

A.4. Detailed Calculation of the Wagner-Whitin Model

```java
public void calculate() {
    Arrays.fill(totalCost, Integer.MAX_VALUE);
    Arrays.fill(orderSchedule, -1);

    for (int j = 0; j < demands.length; j++) {
        costMatrix[0][j] = (j == 0 ? orderCost : costMatrix[0][j - 1]) +
            demands[j] * holdingCostPerUnitPerPeriod * j;
        totalCost[j] = costMatrix[0][j];
        orderSchedule[j] = 0;
    }
    for (int startWeek = 1; startWeek < demands.length; startWeek++) {
        for (int endWeek = startWeek; endWeek < demands.length; endWeek++) {
            int holdingCost = 0;
            for (int week = startWeek; week <= endWeek; week++) {
                holdingCost += demands[week] * holdingCostPerUnitPerPeriod *
                    (week - startWeek);
            }
            int costForThisOrder = orderCost + holdingCost;
            costMatrix[startWeek][endWeek] = costForThisOrder + (startWeek >
                0 ? totalCost[startWeek - 1] : 0);

            if (totalCost[endWeek] > costMatrix[startWeek][endWeek]) {
                totalCost[endWeek] = costMatrix[startWeek][endWeek];
                orderSchedule[endWeek] = startWeek;
            }
        }
    }

    int lastOrderIndex = -1;
    int sum = 0;
    for (int i = 0; i < orderSchedule.length; i++) {
        if (i == 0 || orderSchedule[i] != orderSchedule[lastOrderIndex]) {
            if (lastOrderIndex != -1) {
                productionPeriods[i] = sum;
            }
            lastOrderIndex = i;
            sum = demands[i];
        } else {
            sum += demands[i];
        }
    }
    if (lastOrderIndex != -1) {
        int lastOrderIndex1 = lastOrderIndex+1;
        productionPeriods[lastOrderIndex1] = sum;
    }
    adjustCostMatrix();
    printCostMatrix();
}
```