CHAPTER 1

ABOUT THE COMPANY

1.1 INTRODUCTION

Kristalball is a Bangalore-based startup founded in 2021, dedicated to revolutionizing inventory management and operational efficiency within the hospitality industry through the integration of artificial intelligence (AI), machine learning (ML), and Internet of Things (IoT) technologies[4,5,6]. The company's mission is to combine the power of human and artificial intelligence to create "humane-tech" products that simplify business transactions, improve performance, and enhance the customer experience[3,5,7].

Kristalball's core offering is a state-of-the-art, AI-driven platform designed to make inventory management hands-free and hassle-free, specifically tailored for high-end bars and hospitality businesses[3,4]. Their solutions leverage predictive analytics and smart technology to forecast demand, optimize stock levels, and reduce wastage, thereby maximizing operational efficiency and profitability[4,5]. By connecting "back-of-the-house" operations with "front-of-the-house" customer engagement, Kristalball provides a unique, integrated approach that empowers businesses to make data-driven decisions and deliver superior service[3,4].

The company emphasizes intuitive design and user experience, ensuring that its technology acts as a seamless, non-intrusive enhancement to daily operations[5,7]. Kristalball's vision is to offer easy-to-operate solutions that address everyday challenges while providing greater predictability about future trends, ultimately enabling bars, brands, and customers to benefit from smarter, more connected hospitality experiences[5,7].

Headquartered in Bellandur, Bangalore, Kristalball operates with a small, agile team and follows a B2B business model, offering its services through licensing, subscriptions, and physical commerce[4,6]. The company's innovative approach and

commitment to “humane technology” position it as a forward-thinking leader in the intersection of technology and hospitality

1.2. COMPANY’S TECHSTACK

Kristalball employs a modern and versatile technology stack to deliver robust, scalable, and user-friendly solutions for the hospitality industry. Their stack is carefully chosen to support rapid development, cross-platform compatibility, and efficient data management across all layers of the application.

Frontend Technologies

React Native: Used for developing cross-platform mobile applications, enabling code reuse and consistent user experiences on both iOS and Android devices. React Native leverages a component-based architecture and integrates with libraries like Expo for streamlined development and deployment1.

Next.js: A React-based framework that brings advanced features such as server-side rendering and static site generation to web applications, enhancing performance and SEO. Next.js is used for building scalable, high-performance web interfaces1.

TypeScript: A statically typed superset of JavaScript, TypeScript is used throughout the codebase to improve code reliability, maintainability, and developer productivity by catching errors at compile time4.

JavaScript: The core scripting language for building interactive web and mobile interfaces, ensuring dynamic user experiences and seamless integration with other frontend technologies3.

Tailwind CSS: A utility-first CSS framework that enables rapid UI development with consistent styling. In React Native projects, Tailwind CSS is often integrated via libraries like NativeWind to bring utility classes to mobile development4.

Backend Technologies

Laravel: A robust PHP framework known for its elegant syntax and powerful features, Laravel is used to build and manage web application backends, handle routing, authentication, and database migrations. It supports rapid API development and integrates smoothly with SQL databases1.

Java: Employed for building scalable backend services and middleware components, Java is known for its performance, security, and suitability for enterprise-level applications[5].

Database Layer

SQL (MySQL): Kristalball relies on SQL databases, such as MySQL, for persistent data storage and efficient data retrieval. The team focuses on query optimization using profiling tools (like the EXPLAIN statement), indexing, and query rewriting to ensure high performance and scalability1.

This technology stack allows Kristalball to deliver feature-rich applications that are maintainable, performant, and adaptable to the evolving needs of the hospitality sector. By leveraging these modern tools and frameworks, Kristalball ensures seamless integration between the frontend, backend, and database layers, supporting both web and mobile platforms1[3,4].

CHAPTER 2

FRONTEND DEVELOPMENT

Introduction to JavaScript Development

JavaScript is a versatile and powerful programming language that is widely used for web development. It allows developers to create dynamic and interactive web applications. In the context of frontend development, JavaScript plays a crucial role in building user interfaces (UIs) and enhancing user experience. This chapter will delve into the JavaScript development process, particularly focusing on the implementation of UIs using the React Native framework.

Understanding JavaScript

JavaScript is a high-level, interpreted programming language that conforms to the ECMAScript specification. It is characterized by its event-driven, functional, and imperative programming styles. JavaScript is primarily used for client-side scripting, but it can also be used for server-side development through environments like Node.js.

Key Features of JavaScript

1. **Dynamic Typing**: JavaScript is dynamically typed, meaning variables can hold values of any type and can change types during execution.
2. **First-Class Functions**: Functions in JavaScript are treated as first-class citizens, allowing them to be passed as arguments, returned from other functions, and assigned to variables.
3. **Prototype-Based Inheritance**: JavaScript uses prototypes for inheritance, enabling objects to inherit properties and methods from other objects.
4. **Event-Driven Programming**: JavaScript excels in handling events, making it ideal for creating interactive web applications.

The Role of JavaScript in Frontend Development

Frontend development involves creating the visual and interactive aspects of a web application that users interact with directly. JavaScript is essential in this process as it enables developers to manipulate the Document Object Model (DOM), handle user events, and communicate with backend services.

Key Concepts in Frontend Development

1. **DOM Manipulation**: JavaScript allows developers to access and modify the DOM, which represents the structure of a web page.
2. **Event Handling**: JavaScript can respond to user actions such as clicks, key presses, and mouse movements.
3. **AJAX**: Asynchronous JavaScript and XML (AJAX) enables asynchronous communication with the server, allowing web pages to update dynamically without reloading.

Introduction to React Native

React Native is a popular framework for building mobile applications using JavaScript and React. It allows developers to create native apps for iOS and Android using a single codebase. React Native leverages the power of React, a JavaScript library for building user interfaces, to create mobile apps that are fast, responsive, and visually appealing.

Key Features of React Native

1. **Cross-Platform Development**: Write once, run anywhere. React Native enables developers to build apps for multiple platforms using the same codebase.
2. **Native Components**: React Native uses native components, providing a look and feel that is consistent with the platform.

3. **Hot Reloading**: Developers can see changes in real-time without recompiling the entire app.
4. **Modular Architecture**: React Native promotes a modular architecture, making code easier to maintain and debug.

The JavaScript Development Process in React Native

Setting Up the Development Environment

To start developing with React Native, you need to set up your development environment. This involves installing Node.js, the React Native CLI, and an emulator or physical device for testing.

1. **Install Node.js**: Node.js is required to run JavaScript outside the browser. It also includes npm, the package manager for JavaScript.
2. **Install React Native CLI**: The React Native Command Line Interface (CLI) is used to create and manage React Native projects.
3. **Set Up an Emulator**: You can use Android Studio or Xcode to set up an emulator for testing your app.

Creating a New React Native Project

Once the environment is set up, you can create a new React Native project using the CLI. This involves running a command that initializes a new project with the necessary files and dependencies.

```
npx react-native init MY-PROJECT
```

Building User Interfaces

React Native uses a component-based architecture, where UIs are built using reusable components. Components can be functional or class-based, and they encapsulate the logic and presentation of a part of the UI.

1. **Functional Components**: These are simple JavaScript functions that return JSX, a syntax extension that looks similar to HTML.

2. **Class Components**: These are ES6 classes that extend React.Component and include lifecycle methods.

Styling Components

React Native uses a styling system similar to CSS but with a few differences. Styles are defined using JavaScript objects and applied to components using the style prop.

```
const styles = {
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  text: {
    fontSize: 20,
    textAlign: 'center',
    margin: 10,
  },
};

const MyComponent = () => (
  <View style={styles.container}>
    <Text style={styles.text}>Hello, React Native!</Text>
    </View>
```

State Management

State management is crucial in React Native applications to handle dynamic data and user interactions. React Native uses React's state and props system to manage state within components.

1. **State**: State is a JavaScript object that holds dynamic data. It is managed within a component and can be updated using the setState method.
2. **Props**: Props are read-only attributes passed from parent components to child components. They allow data to flow through the component hierarchy.
3. **Context API**: The Context API is used to share state across multiple components without passing props manually at every level. It is particularly useful for global state management, such as theme settings or user authentication status.
4. **Hooks**: React Native leverages hooks like useState, useEffect, and useContext to manage state and side effects in functional components. Hooks provide a more concise and readable way to handle state compared to class components.
5. **Redux**: Redux is a popular state management library that provides a predictable state container for JavaScript apps. It helps manage complex state logic by centralizing state in a single store and using actions and reducers to update the state.
6. **MobX**: MobX is another state management library that simplifies state management by using observable data structures and reactions. It allows for automatic updates to the UI when the state changes, making it easier to manage complex state dependencies.
7. **Local State vs. Global State**: Local state is managed within individual components using hooks like useState, while global state is managed across the entire application using libraries like Redux or Context API. Understanding the difference helps in deciding the appropriate state management approach.
8. **Lifecycle Methods**: In class components, lifecycle methods such as componentDidMount, componentDidUpdate, and componentWillUnmount are used to manage state and side effects. In functional components, hooks like useEffect serve a similar purpose.
9. **Performance Optimization**: Efficient state management can improve application performance. Techniques such as memoization (React.memo), lazy loading, and avoiding unnecessary re-renders help optimize state updates and enhance user experience.
10. **State Persistence**: Persisting state across sessions can be achieved using libraries like redux-persist or AsyncStorage. This ensures that the state is retained even after the application is closed and reopened.

11. **Error Handling**: Proper error handling in state management ensures that the application remains robust and user-friendly. Techniques include validating state updates, using try-catch blocks, and displaying user-friendly error messages.
12. **Testing State Management**: Testing state management involves writing unit tests and integration tests to ensure that state updates and interactions work as expected. Tools like Jest and React Testing Library are commonly used for this purpose.

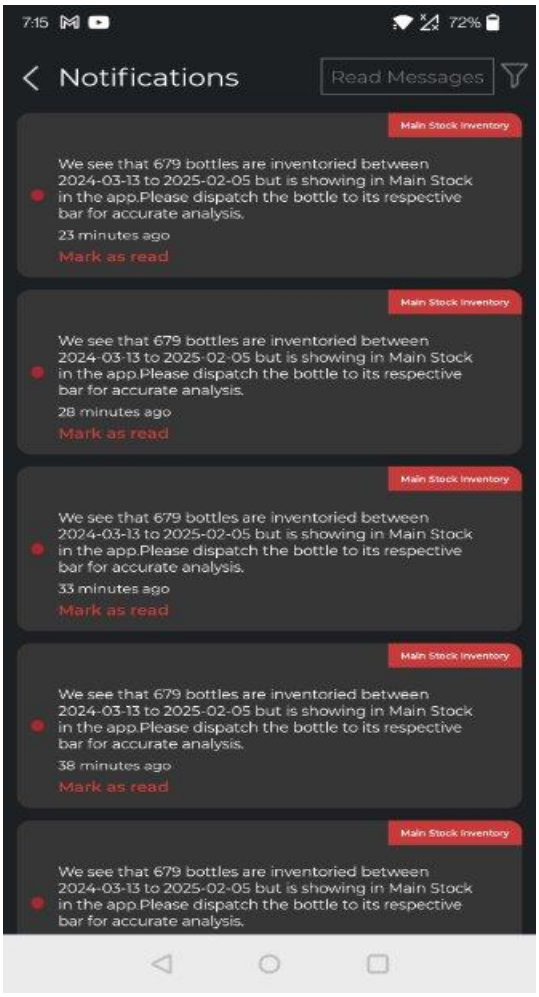


FIGURE 1: Notification Menu

Navigation

Navigation is an essential aspect of mobile applications. React Native provides several libraries for navigation, such as react-navigation, which allows developers to implement various navigation patterns like stack, tab, and drawer navigation.

```
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

const Stack = createStackNavigator();

const App = () => (
  <NavigationContainer>
    <Stack.Navigator>
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="Details" component={DetailsScreen}
      <Stack.Screen name="Contact" component={ContactScreen} />
            <Stack.Screen name="Contact" component={AboutScreen} />
    </Stack.Navigator>
  </NavigationContainer>
);
```

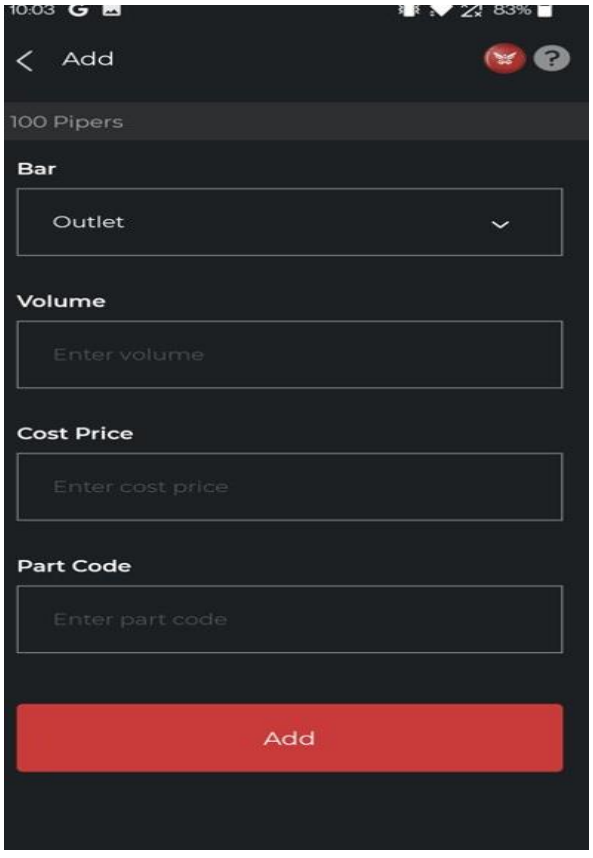


FIGURE 2: FORM PAGE

Introduction to Next.js

Next.js is a popular framework for building server-rendered React applications. It provides a robust set of features for creating high-performance web applications, including static site generation, server-side rendering, and API routes. Next.js simplifies the development process by offering a comprehensive solution for building modern web applications.

Key Features of Next.js

1. Server-Side Rendering (SSR): Next.js allows pages to be rendered on the server, improving performance and SEO.
2. Static Site Generation (SSG): Next.js can generate static HTML pages at build time, which can be served to users quickly.
3. API Routes: Next.js provides a way to create API endpoints within the same project, simplifying backend integration.
4. Automatic Code Splitting: Next.js automatically splits code, ensuring that only the necessary code is loaded for each page.

The JavaScript Development Process in Next.js

Setting Up the Development Environment

To start developing with Next.js, you need to set up your development environment. This involves installing Node.js and creating a new Next.js project.

1. Install Node.js: Node.js is required to run JavaScript outside the browser. It also includes npm, the package manager for JavaScript.
2. Create a New Next.js Project: You can create a new Next.js project using the create-next-app command, which sets up a project with the necessary files and dependencies.

Building User Interfaces

Next.js uses React for building user interfaces. Components in Next.js are built using React's component-based architecture, where UIs are constructed using reusable components.

1. Functional Components: These are simple JavaScript functions that return JSX, a syntax extension that looks similar to HTML.
2. Class Components: These are ES6 classes that extend React.Component and include lifecycle methods.

Styling Components

Next.js supports various styling options, including CSS, Sass, and styled-components. Styles can be defined using CSS modules, which scope styles locally to components, or using global styles.

State Management

State management is crucial in Next.js applications to handle dynamic data and user interactions. Next.js uses React's state and props system to manage state within components.

1. State: State is a JavaScript object that holds dynamic data. It is managed within a component and can be updated using the useState hook.
2. Props: Props are read-only attributes passed from parent components to child components. They allow data to flow through the component hierarchy.

Data Fetching

Next.js provides several methods for fetching data, including getStaticProps, getServerSideProps, and getInitialProps. These methods allow developers to fetch data at build time, on the server, or on the client.

1. getStaticProps: Fetches data at build time for static site generation.
2. getServerSideProps: Fetches data on the server for server-side rendering.
3. getInitialProps: Fetches data on the client for client-side rendering.

Routing

Next.js uses a file-based routing system, where the file structure of the pages directory defines the routes of the application. Each file in the pages directory corresponds to a route.

JavaScript development, particularly with Next.js, offers a robust and efficient way to build modern web applications. By leveraging the power of JavaScript and the Next.js framework, developers can create high-quality, performant apps that provide excellent user experiences. This chapter has provided an overview of the JavaScript development process, from setting up the environment to building and styling components, managing state, fetching data, and implementing routing.

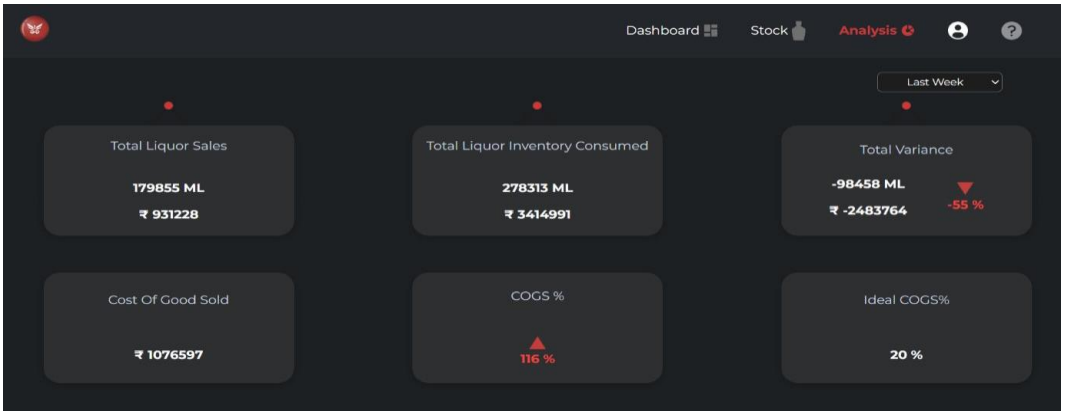


FIGURE 3: WEBSITE ANALYSIS PAGE

CHAPTER 3

BACKEND DEVELOPMENT USING LARAVEL

3.1 Introduction

Laravel is a powerful and elegant PHP framework designed for web application development. It follows the Model-View-Controller (MVC) architectural pattern, which helps in organizing and structuring code in a clean and maintainable way. Laravel provides a rich set of features, including routing, authentication, sessions, and caching, making it a popular choice for backend development.

3.2 MVC Architecture

The MVC architecture is a design pattern that separates an application into three main components: Model, View, and Controller. This separation helps in managing the complexity of large applications by dividing the responsibilities among different parts of the application.

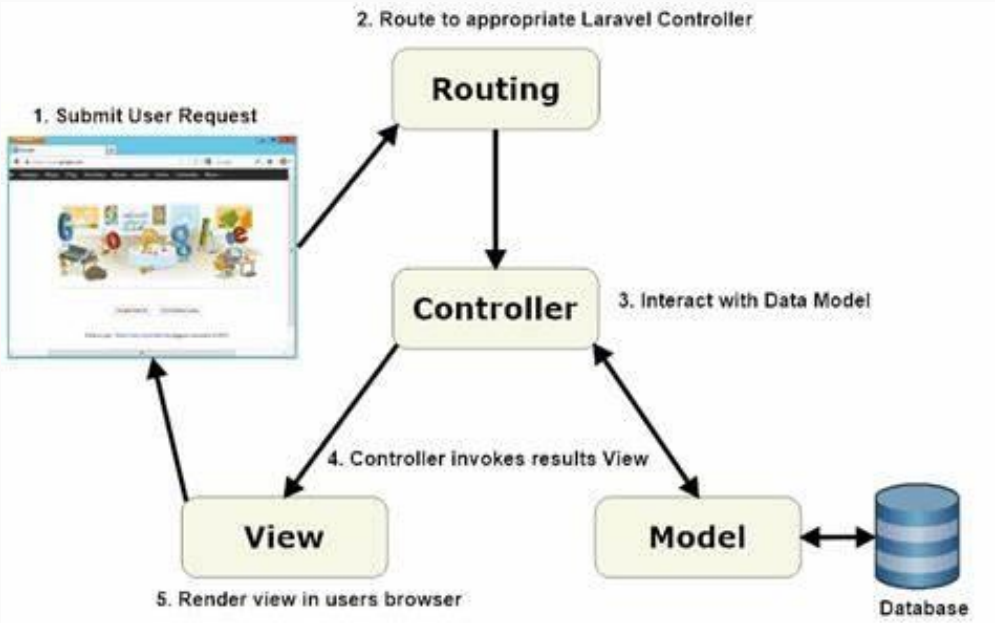


FIGURE 4: MVC ARCHITECTURE

Model

The Model represents the data and the business logic of the application. It is responsible for retrieving data from the database, processing it, and returning it to the controller.

In Laravel, models are typically defined as Eloquent ORM classes, which provide an easy and intuitive way to interact with the database.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Post extends Model

{

    protected $fillable = ['title', 'content'];

}
```

View

The View is responsible for displaying the data to the user. It is the presentation layer of the application. In Laravel, views are typically written in Blade, a simple yet powerful templating engine that allows developers to create dynamic HTML content.

```
<!-- resources/views/posts/index.blade.php -->
@extends('layouts.app')


@section('content')
    <h1>Posts</h1>
    @foreach ($posts as $post)
        <div>
            <h2>{{ $post->title }}</h2>
            <p>{{ $post->content }}</p>
        </div>
    @endforeach
@endsection
```

Controller

The Controller acts as an intermediary between the Model and the View. It handles user input, processes it, and returns the appropriate response. In Laravel, controllers are typically defined as classes that extend the base Controller class.

```
namespace App\Http\Controllers;

use App\Models\Post;

use Illuminate\Http\Request;

class PostController extends Controller

{

    public function index()

    {

        $posts = Post::all();

        return view('posts.index', compact('posts'));

    }

    public function create()

    {

        return view('posts.create');

    }

    public function store(Request $request)

    {

        $post = Post::create($request->all());

        return redirect()->route('posts.index');
```

```
    }
}
```

3.3 CRUD Operations

CRUD stands for Create, Read, Update, and Delete. These are the basic operations that can be performed on data in a database. Laravel provides a simple and elegant way to perform these operations using MVC design.

Create

It involves adding new records to the database. In Laravel, this can be done using Eloquent's create method.

```
public function store(Request $request)
{
    $post = Post::create($request->all());
    return redirect()->route('posts.index');
}
```

Read

The Read operation involves retrieving records from the database. In Laravel, this can be done using Eloquent's all method or query builder methods.

```
public function index()
{
    $posts = Post::all();
    return view('posts.index', compact('posts'));
}
```

Update

The Update operation involves modifying existing records in the database. In Laravel, this can be done using Eloquent's update method.

```
public function update(Request $request, $id)

{

    $post = Post::findOrFail($id);

    $post->update($request->all());

    return redirect()->route('posts.index');

}
```

Delete

The Delete operation involves removing records from the database. In Laravel, this can be done using Eloquent's delete method.

```
public function update(Request $request, $id)

{

    $post = Post::findOrFail($id);

    $post->update($request->all());

    return redirect()->route('posts.index');

}
```

Backend development with Laravel offers a robust and efficient way to build modern web applications. By leveraging the power of the MVC architecture and Eloquent ORM, developers can create scalable and maintainable applications with ease. This chapter has provided an overview of the Laravel framework, the MVC architecture, and the implementation of CRUD operations.

CHAPTER 4

SQL OPTIMIZATION

4.1 SQL Engine Architecture

SQL (Structured Query Language) is the standard language for managing and manipulating relational databases. Understanding the internal workings of an SQL engine is crucial for optimizing database performance. The SQL engine is responsible for executing SQL queries and consists of several components that work together to process and retrieve data efficiently.

Components of SQL Engine

Component	**Description**
Parser	Checks the syntax of SQL queries and converts them into an internal format.
Optimizer	Analyzes parsed queries and generates the most efficient execution plan.
Executor	Executes the plan generated by the optimizer by interacting with the storage engine.
Storage Engine	Manages data files, indexes, and other storage structures for efficient data access.

TABLE 1: COMPONENTS OF SQL

SQL Query Execution Process

1. **Parsing**: The SQL query is parsed to check for syntax errors and convert it into an internal representation.
2. **Optimization**: The optimizer analyzes the query and generates an execution plan.
3. **Execution**: The executor executes the plan by interacting with the storage engine to retrieve and manipulate data.
4. **Result**: The results are returned to the client application.

4.2 Different Optimization Techniques

Optimizing SQL queries is essential for improving database performance and ensuring efficient data retrieval. There are several techniques that can be employed to optimize SQL queries, including indexing, profiling, and query rewriting.

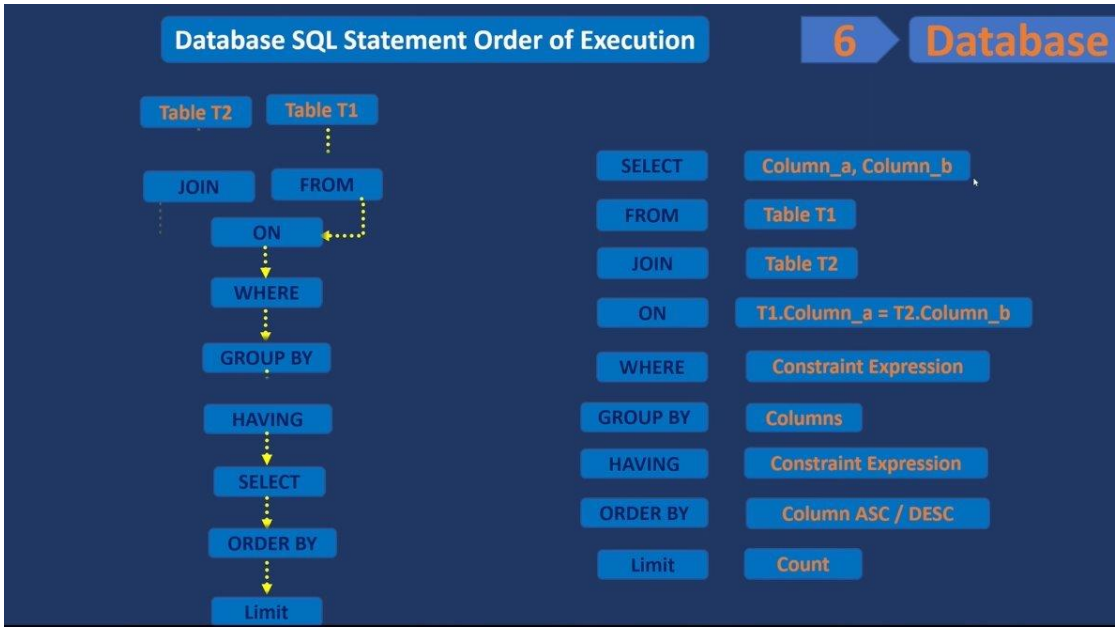


FIGURE 5:SQL STATEMENT ORDER OF EXECUTION

Indexing

Indexes are data structures that improve the speed of data retrieval operations on a database table. They work by providing quick access to rows based on the values of indexed columns. Proper indexing can significantly enhance query performance.

TABLE 2: TYPES OF INDEXES

Index Type	**Description**
Primary Index	Automatically created on the primary key of a table.
Unique Index	Ensures that all values in the indexed column are unique.
Composite Index	Created on multiple columns to optimize queries that filter based on multiple criteria.

1. **Creating Indexes**: Indexes can be created on one or more columns of a table. The choice of columns to index depends on the queries that are frequently executed.

CREATE INDEX idx_post_title ON posts(title);

2. **Index Maintenance**: Indexes need to be maintained and updated as data in the table changes. Regular maintenance ensures that indexes remain efficient.

```
mysql> EXPLAIN SELECT id, title, production_year FROM movies
       WHERE title = 'Bambi' ORDER BY production_year\G
************* 1. row *************
           id: 1
  select_type: SIMPLE
        table: movies
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 3331824
     filtered: 10.00
        Extra: Using where; Using filesort
1 row in set, 1 warning (0.00 sec)
```

```
mysql> ALTER TABLE movies ADD INDEX (title(30));
Query OK, 0 rows affected (8.09 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> EXPLAIN SELECT id, title, production_year FROM movies
       WHERE title = 'Bambi' ORDER by production_year\G
************* 1. row *************
           id: 1
  select_type: SIMPLE
        table: movies
         type: ref
possible_keys: title
          key: title
      key_len: 52
          ref: const
         rows: 4
     filtered: 100.00
        Extra: Using where; Using filesort
1 row in set, 1 warning (0.00 sec)
```

FIGURE 6: INDEXING

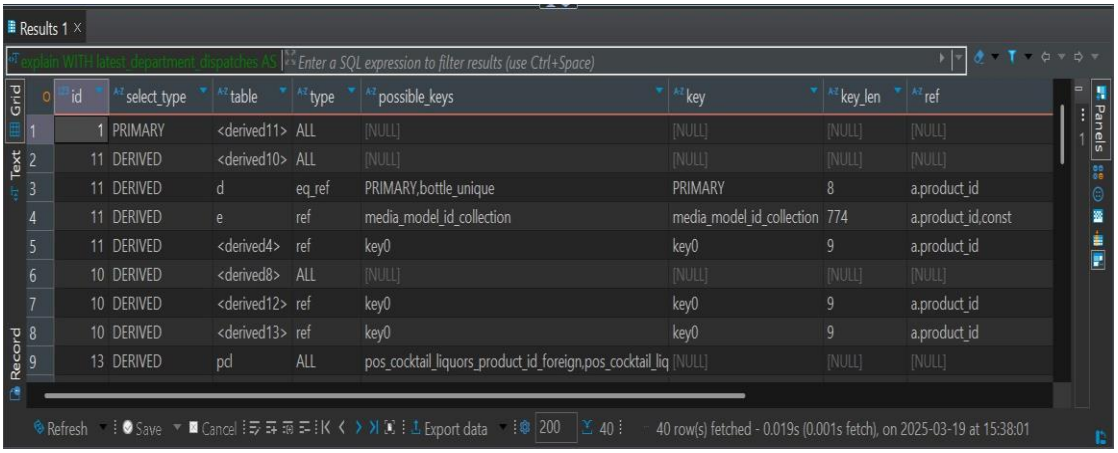
Profiling

Profiling involves analyzing the performance of SQL queries to identify bottlenecks and areas for improvement. Tools like the SQL Profiler can be used to monitor query execution and gather performance metrics.

Profiling Metric	Description
Query Execution Time	Measure the time taken to execute queries and identify slow-running queries.
Resource Usage	Monitor CPU, memory, and I/O usage during query execution.
Execution Plan Analysis	Examine the execution plans generated by the optimizer to identify inefficient operations.

TABLE 3: PROFILING METRIC

```
mysql> EXPLAIN SELECT * FROM sakila.film_actor WHERE film_id = 1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: film_actor
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 5073
        Extra: Using where
```



	id	select_type	table	type	possible_keys	key	key_len	ref
1	1	PRIMARY	<derived11>	ALL	[NULL]	[NULL]	[NULL]	[NULL]
2	11	DERIVED	<derived10>	ALL	[NULL]	[NULL]	[NULL]	[NULL]
3	11	DERIVED	d	eq_ref	PRIMARY,bottle_unique	PRIMARY	8	a.product_id
4	11	DERIVED	e	ref	media_model_id_collection	media_model_id_collection	774	a.product_id,const
5	11	DERIVED	<derived4>	ref	key0	key0	9	a.product_id
6	10	DERIVED	<derived8>	ALL	[NULL]	[NULL]	[NULL]	[NULL]
7	10	DERIVED	<derived12>	ref	key0	key0	9	a.product_id
8	10	DERIVED	<derived13>	ref	key0	key0	9	a.product_id
9	13	DERIVED	pcl	ALL	pos_cocktail_liquors_product_id_foreign,pos_cocktail_liq	[NULL]	[NULL]	[NULL]

FIGURE 7: PROFILING

Query Rewriting

Query rewriting involves modifying SQL queries to improve their performance. This can include simplifying complex queries, using joins efficiently, and avoiding unnecessary subqueries.

1. **Simplifying Queries**: Break down complex queries into simpler ones that are easier to optimize.

Complex query

```
SELECT * FROM posts WHERE title LIKE '%optimization%' AND content LIKE '%performance%';
```

Simplified query

```
SELECT * FROM posts WHERE title LIKE '%optimization%' UNION SELECT * FROM posts WHERE content LIKE '%performance%';
```

2. **Using Joins Efficiently**: Use appropriate join types (INNER JOIN, LEFT JOIN, etc.) and ensure that join conditions are optimized.

```
SELECT posts.title, comments.content
FROM posts
INNER JOIN comments ON posts.id = comments.post_id;
```

3. **Avoiding Subqueries**: Replace subqueries with joins or other techniques to improve performance.

Subquery

```
SELECT * FROM posts WHERE id IN (SELECT post_id FROM comments
WHERE content LIKE '%optimization%');
```

Join

```
SELECT posts.*
FROM posts
INNER JOIN comments ON posts.id = comments.post_id
WHERE comments.content LIKE '%optimization%';
```

SQL optimization is a critical aspect of database management that ensures efficient data retrieval and overall system performance. By understanding the internal workings of the SQL engine and employing various optimization techniques such as indexing, profiling, and query rewriting, developers can significantly enhance the performance of their SQL queries.

CHAPTER 5

TIME SERIES ANALYSIS

Introduction to Time Series Analysis

Time Series Analysis is a statistical technique that deals with analyzing time-ordered data points. It is used to identify patterns, trends, and seasonal variations in data over time. In the context of predicting the par level of alcohol bottles each month, Time Series Analysis helps in forecasting future inventory requirements based on historical data.

Understanding Time Series Data

Time Series data is a sequence of data points collected or recorded at specific time intervals. These intervals can be hourly, daily, weekly, monthly, or yearly. Time Series Analysis involves several key components:

Univariate Analysis

Univariate analysis involves the examination of a single variable. It is the simplest form of data analysis and is used to describe and summarize data. The primary goal of univariate analysis is to understand the distribution and characteristics of the variable in question. This type of analysis includes:

- **Descriptive Statistics**: Measures such as mean, median, mode, variance, and standard deviation.
- **Frequency Distribution**: The count of occurrences of each value or range of values.
- **Graphical Representation**: Visual tools like histograms, bar charts, and box plots to illustrate the distribution of the variable.

Multivariate Analysis

Multivariate analysis involves the examination of more than one variable at a time. It is used to understand relationships between variables and to model complex phenomena. This type of analysis can reveal patterns and interactions that are not apparent in univariate analysis. Key techniques in multivariate analysis include:

- **Multiple Regression**: Examines the relationship between one dependent variable and multiple independent variables.

- **Principal Component Analysis (PCA)**: Reduces the dimensionality of data by transforming it into a set of uncorrelated variables called principal components.
- **Factor Analysis**: Identifies underlying relationships between variables by grouping them into factors.
- **Cluster Analysis**: Groups similar observations into clusters based on their characteristics.
- **Multivariate Analysis of Variance (MANOVA)**: Extends ANOVA to multiple dependent variables to understand if changes in independent variables have significant effects on them.

Multivariate analysis is widely used in fields such as finance, marketing, social sciences, and biology to analyze complex data sets and make informed decisions based on multiple variables.

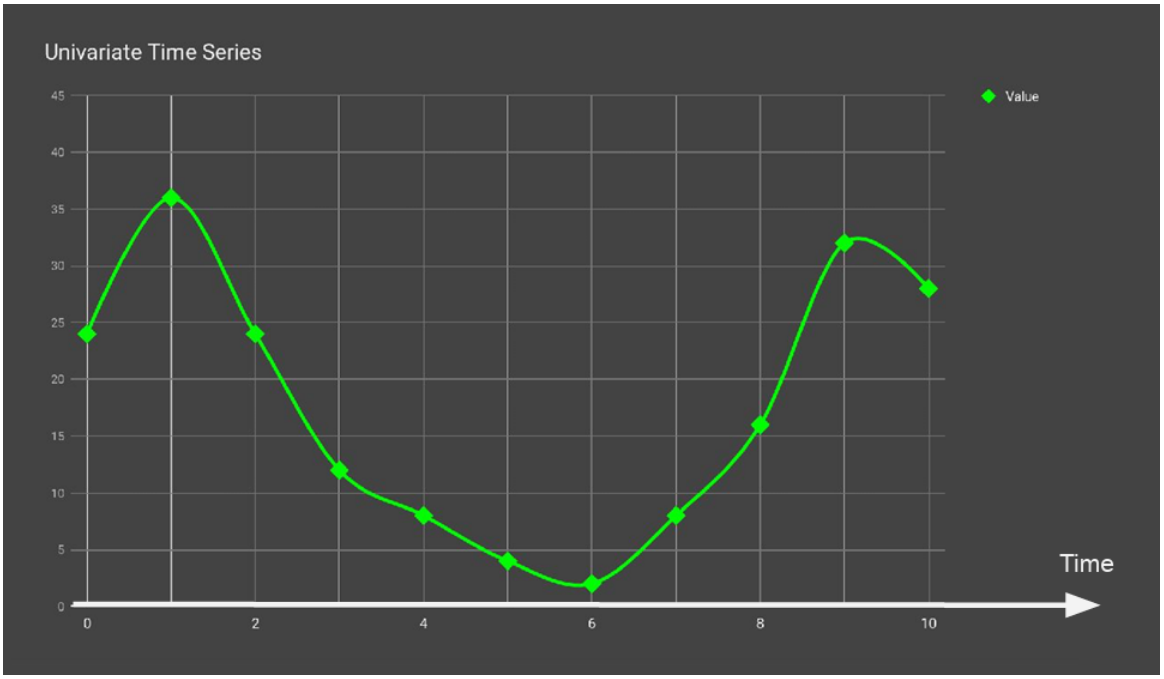


FIGURE 8 UNIVARIATE TIME SERIES

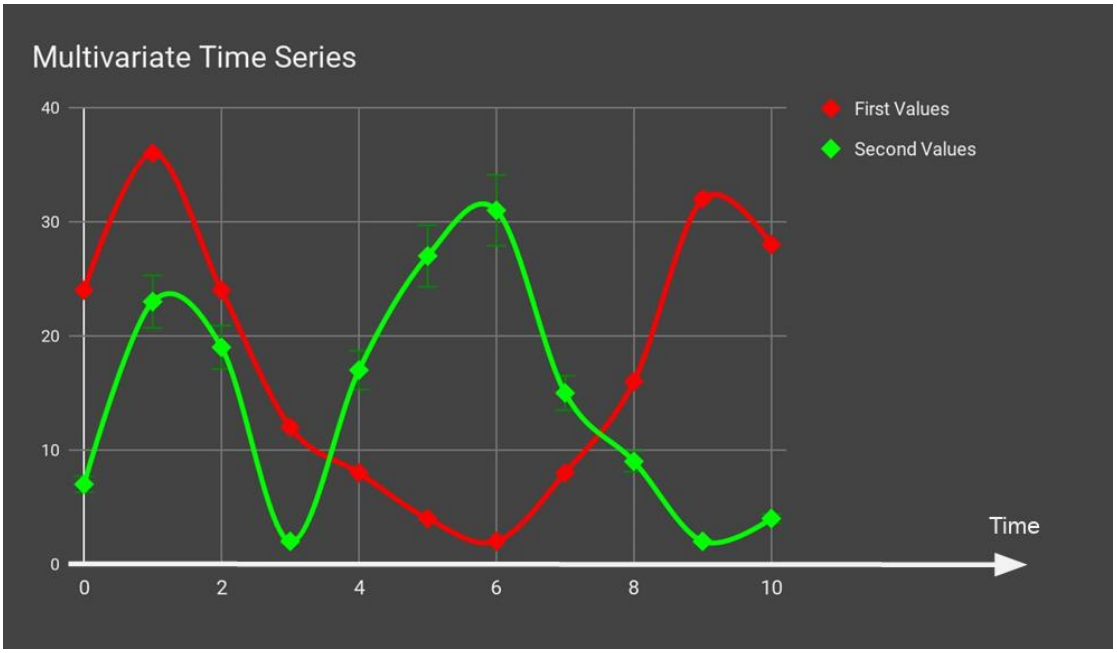


FIGURE 9 MULTIVARIATE TIME SERIES

Components of Time Series Data

Component	Description
Trend	Long-term movement in the data over a period of time.
Seasonality	Regular, repeating patterns or cycles in the data.
Noise	Random variations that do not follow a pattern.

TABLE 4: COMPONENTS OF TIME SERIES DATA

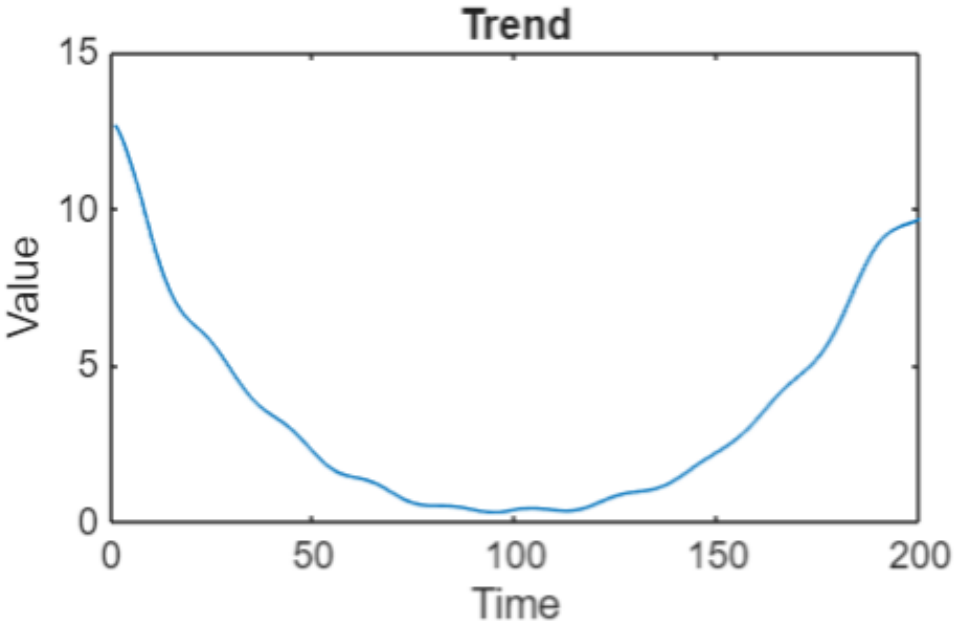


FIGURE 10 TREND

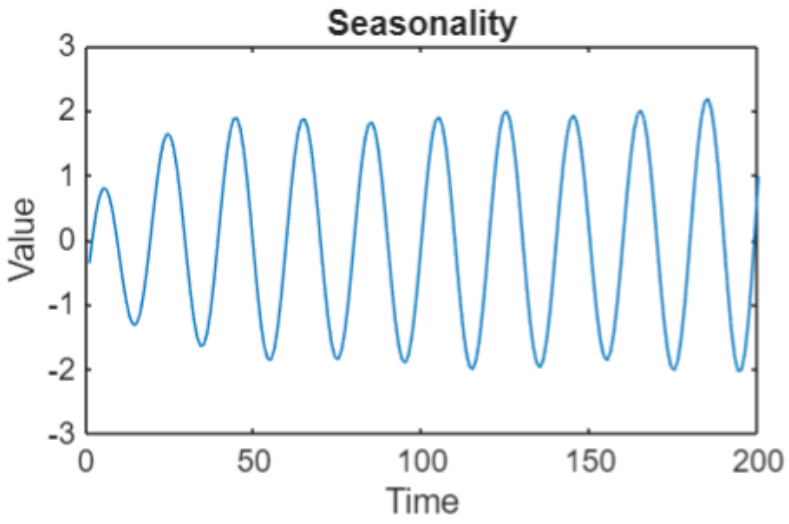


FIGURE 11 SEASONALITY

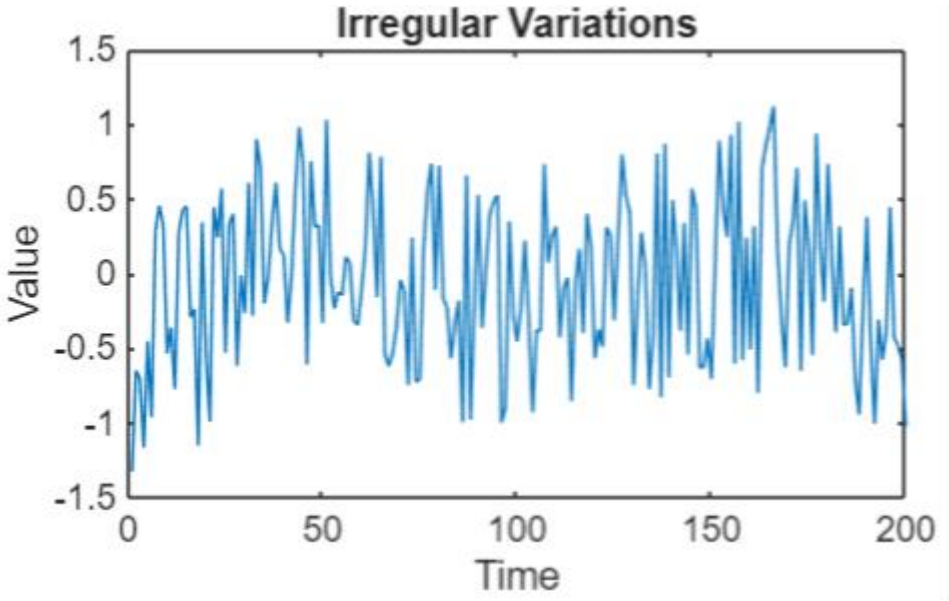


FIGURE 12 NOISE

Predicting Par Level of Alcohol Bottles

Predicting the par level of alcohol bottles each month involves analyzing historical sales data to forecast future inventory needs. This ensures that the right amount of stock is maintained to meet customer demand without overstocking or understocking.

Steps in Time Series Analysis

1. **Data Collection**: Gather historical sales data of alcohol bottles.
2. **Data Preprocessing**: Clean and prepare the data for analysis.
3. **Exploratory Data Analysis (EDA)**: Analyze the data to identify patterns, trends, and seasonality.
4. **Model Selection**: Choose appropriate time series models for forecasting.
5. **Model Training**: Train the selected models using historical data.
6. **Forecasting**: Use the trained models to predict future par levels.
7. **Evaluation**: Assess the accuracy of the forecasts and refine the models if necessary.

Data Collection

Collecting historical sales data is the first step in Time Series Analysis. This data should include the number of alcohol bottles sold each month over a significant period.

Month	Bottles Sold
January	1200
February	1100
March	1300
April	1250
May	1400
June	1350
July	1500
August	1450
September	1600
October	1550
November	1700
December	1650

TABLE 5:EXAMPLE DATASET

Data Preprocessing

Data preprocessing involves cleaning the data to remove any inconsistencies, missing values, or outliers. This step ensures that the data is ready for analysis.

Exploratory Data Analysis (EDA)

EDA involves visualizing the data to identify patterns, trends, and seasonality. Common techniques include plotting the data and calculating summary statistics.

Month	Bottles Sold	Moving Average
January	1200	-
February	1100	-
March	1300	-
April	1250	-
May	1400	1250
June	1350	1300
July	1500	1375
August	1450	1425
September	1600	1475
October	1550	1525
November	1700	1575
December	1650	1625

TABLE 6: MOVING AVERAGE

```
errors = forecasts - actual

mse = np.square(errors).mean()

rmse = np.sqrt(mse)

mae = np.abs(errors).mean()

mape = np.abs(errors / x_valid).mean()
```

FIGURE 13: FORMULAS

Model Selection

Selecting the appropriate time series models is crucial for accurate forecasting. Common models include:

1. **ARIMA (AutoRegressive Integrated Moving Average)**: Combines autoregression, differencing, and moving average components.
2. **Exponential Smoothing**: Uses weighted averages of past observations to forecast future values.
3. **Seasonal Decomposition**: Separates the time series into trend, seasonal, and residual components.

Model Training

Training the selected models involves fitting them to the historical data. This step requires splitting the data into training and testing sets to evaluate model performance.

Forecasting

Using the trained models, forecasts are generated for future par levels of alcohol bottles. These forecasts help in planning inventory requirements.

TABLE 6: FORECAST DATA

Month	Forecasted Bottles Sold
January	1700
February	1650
March	1800
April	1750
May	1900
June	1850
July	2000
August	1950
September	2100
October	2050
November	2200
December	2150

Evaluation

Evaluating the accuracy of the forecasts involves comparing the predicted values with actual sales data. Metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are commonly used.

Month	Actual Bottles Sold	Forecasted Bottles Sold	Error
January	1700	1700	0
February	1600	1650	50
March	1850	1800	50
April	1750	1750	0
May	1950	1900	50
June	1800	1850	50
July	2050	2000	50
August	1900	1950	50
September	2150	2100	50
October	2000	2050	50
November	2250	2200	50
December	2100	2150	50

TABLE 7: FORECAST DATA WITH ERROR

CHAPTER 6

CONCLUSION AND FUTURE SCOPE OF WORK

The internship experience at Kristalball and Cognizant has been immensely valuable, providing a comprehensive understanding of full stack development and performance optimization. Throughout the internship, I had the opportunity to work on various aspects of web and mobile application development, including frontend technologies like React Native and Next.js, backend frameworks such as Laravel, and database management with MySQL.

Key takeaways from this internship include:

1. Frontend Development: I gained proficiency in implementing responsive UI components, debugging, and optimizing web applications. The use of React Native and Next.js allowed me to create seamless user experiences across different platforms.

2. Backend Development: Working with Laravel, I developed and integrated new features, ensuring robust validation and error handling. This experience enhanced my ability to manage backend services and middleware components efficiently.

3. Database Optimization: Conducting SQL profiling and applying indexing and query rewriting techniques significantly improved application performance. This knowledge is crucial for developing scalable and high-performance applications.

4. Time Series Analysis: Performing time series analysis on inventory data provided insights into predicting future inventory requirements, which is essential for efficient stock management in the hospitality sector.

Overall, this internship has deepened my understanding of full stack development, performance optimization, and data analysis. It has equipped me with the skills to deliver scalable, user-centric solutions in a fast-paced startup environment. The hands-on experience and the guidance from my mentors have been invaluable in shaping my technical abilities and professional growth.

Future Scope

The work done during this internship opens up several avenues for future exploration and improvement:

1. Enhanced UI/UX Design: Further refining the user interface and user experience to make applications more intuitive and engaging for users.

2. Advanced Backend Features: Implementing more complex backend functionalities, such as real-time data processing and advanced security measures.

3. Database Management: Exploring more sophisticated database optimization techniques and tools to handle larger datasets and improve query performance.

4. Predictive Analytics: Leveraging machine learning algorithms for more accurate inventory forecasting and demand prediction.

5. Scalability: Ensuring that applications can scale efficiently to handle increased user load and data volume.

By continuing to build on the foundation laid during this internship, I aim to contribute to the development of innovative and impactful technology solutions in the future.

REFERENCES

1. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/59164806/bf1b0416-a1c6-4a28-a518-58eb96c1422c/mid.pdf
2. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/59164806/d2565b91-5fc4-4d19-b4e3-33f9c68b1994/mid-ppt.pptx
3. https://www.linkedin.com/pulse/full-stack-php-developers-vs-front-end-back-whats-difference-sapra-eamzc
4. https://www.unleashedsoftware.com/blog/par-levels-in-inventory-management-with-formula-examples/
5. https://talent500.com/blog/full-stack-developer-guide/
6. https://www.netsuite.com/portal/resource/articles/inventory-management/inventory-forecasting.shtml
7. https://talentonlease.com/blogs/important-skills-of-a-full-stack-developer/
8. https://www.inventory-planner.com/ultimate-guide-to-inventory-forecasting/
9. https://www.samarpaninfotech.com/blog/guide-to-full-stack-development/
10. https://preset.io/blog/time-series-forecasting-a-complete-guide/
11. https://www.youtube.com/watch?v=pKtyTLARndk
12. https://www.influxdata.com/time-series-forecasting-methods/
13. https://cran.r-project.org/web/packages/forecast/forecast.pdf
14. https://www.advancinganalytics.co.uk/blog/2021/06/22/10-incredibly-useful-time-series-forecasting-algorithms
15. https://intelliarts.com/blog/time-series-analysis-examples/
16. https://www.linkedin.com/advice/0/what-best-ways-evaluate-full-stack-developers-ability-eyohf
17. https://www.linkedin.com/advice/0/what-best-ways-evaluate-full-stack-developers-ability-eyohf
18. https://www.linkedin.com/advice/0/what-best-ways-evaluate-full-stack-developers-ability-eyohf
19. https://www.linkedin.com/advice/0/what-best-ways-evaluate-full-stack-developers-ability-eyohf
20. https://www.linkedin.com/advice/0/what-best-ways-evaluate-full-stack-developers-ability-eyohf