A REPORT

ON

CARGOES SHIELD

By

Name of the student	**Registration No.**
TANVI BAHEDIA	20BCE1667

Prepared in the partial fulfilment of the

Industrial Internship course

AT

DP World – Bangalore

Address: Torrey Pines Building, 4th Floor, Embassy Golf Links Business Park, Bengaluru, Karnataka 560071

Global leader in port operations and logistics.

(4th May 2023 – 4th July 2023)

JOINING REPORT

Date: 04-05-2023

Name of the Student	Tanvi Bahedia
Name and Address of the Organization	**DP WORLD** Address: Torrey Pines Building, 4th Floor, Embassy Golf Links Business Park Bengaluru, Karnataka 560071
Start Date	04-05-2023
End Date	04-07-2023
Location of the Project	Karnataka
Name and Designation of the Industry Guide/ Industry Mentor for the Project	Ritesh Singh Director Quality Assurance and Automation Sai Anudeep Adimulapu Lead SDET
Mentors E-mail Address	ritesh.singh@dpworld.com sai.adimulapu@dpworld.com

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude and appreciation to the mentors at **DP World** for providing me with an invaluable internship opportunity. Their guidance, support, and expertise have been instrumental in shaping my professional growth and allowing me to gain hands-on experience in the industry.

Throughout my internship, my mentors, **Sai Anudeep Adimulapu** and **Ritesh Singh** at **DP World** have consistently demonstrated a commitment to my development and have gone above and beyond to ensure that I had a meaningful and enriching experience. They have generously shared their knowledge, insights, and best practices, allowing me to learn from their expertise and acquire new skills.

Their guidance and mentorship have not only provided me with a solid foundation in the field but have also instilled in me a deep passion for the work and a desire to excel. Their patience, encouragement, and willingness to answer my questions have made me feel supported and valued as a member of the team.

Finally, I would also like to express my gratitude to **DP World** as an organization for providing such a supportive and inclusive environment for interns. The opportunities they offer and the commitment they demonstrate to fostering talent are truly commendable.

Once again, I extend my heartfelt appreciation to all my mentors at **DP World** for their guidance, expertise, and belief in my abilities. I am truly grateful for the opportunity to have learned from them and will always carry their influence with me as I continue to grow in my career.

ABSTRACT

This abstract provides an overview of a cargo insurance product developed by DP World, leveraging various technologies and frameworks for different aspects of the system. The backend of the application is built using Node.js, Express, and Nest.js frameworks. These frameworks offer a robust foundation for developing scalable and efficient server-side applications. TypeScript is used as the programming language, which provides static typing and improved developer productivity. The frontend of the system is developed using Svelte, a modern JavaScript framework, along with Tailwind CSS for styling and UI components. This combination allows for the creation of dynamic and responsive user interfaces, providing an intuitive experience for cargo insurance management.

The data layer of the application utilizes Prisma, an open-source database toolkit, along with PostgreSQL as the database management system. This combination offers efficient data storage, retrieval, and management capabilities, ensuring the reliability and security of cargo insurance-related information. To support the development process, Azure DevOps is employed for continuous integration and deployment. It provides a comprehensive set of tools and services for automating the build, test, and deployment processes, ensuring smooth and efficient software delivery. Redis, an in-memory data structure store, is integrated into the system to enable caching of frequently accessed data, improving performance and response times. To ensure the reliability and performance of the application under different load scenarios, K6, a modern load testing tool, is employed. It allows for simulating various user interactions and stress testing the system.

Code quality and security are addressed through the use of SonarQube, an open-source code quality platform. It provides continuous code inspection, identifying potential vulnerabilities, and maintaining code quality standards. Cobertura, a code coverage tool, is utilized to measure the extent of code coverage achieved by the test suite, enabling developers to assess the effectiveness of their tests.

For deployment and management of the application in a cloud environment, Kubernetes, an open-source container orchestration platform, is utilized. This allows for efficient scaling, load balancing, and high availability of the cargo insurance product. Argus deployment is used to facilitate secure and efficient deployment of the cargo insurance application, ensuring a smooth rollout and minimizing potential disruptions.

By leveraging these technologies and tools, DP World can develop a comprehensive and robust cargo insurance product. The system provides a reliable backend, user-friendly frontend, efficient data management, and employs industry-standard practices for continuous integration, deployment, load testing, code quality analysis, and security.

TABLE OF CONTENTS

JOINING REPORT	i
ACKNOWLEDGEMENT	ii
ABSTRACT	iii
TABLE OF CONTENTS	iv
1. A BRIEF INTRODUCTION OF THE ORGANIZATION’S BUSINESS SECTOR	1
2. OVERVIEW OF THE ORGANIZATION	2
3. INTERNSHIP EXPERIENCE	3
4. CHALLENGES FACED	3
5. INTRODUCTION	4
5.1 Project Introduction	
5.2 Motivation	
5.3 Goal of the Project	
6. ACCOMPLISHMENTS	5
7. USER-FLOW	12
8. TECHNICAL OVERVIEW	13
9. ARCHITECTURE OVERVIEW	15
10. CONCLUSION	17
10. GOALS & AMBITIONS	17
LETTER OF COMPLETION	
CERTIFICATE	

A BRIEF INTRODUCTION OF THE ORGANIZATION’S BUSINESS SECTOR

DP World is a leading global provider of smart logistics solutions, specializing in port operations and trade-enabling services. The organization operates in the business sector of port and terminal operations, serving as a crucial link in the global supply chain. It’s core business revolves around managing and operating container terminals in strategic locations worldwide. The company plays a vital role in facilitating international trade by handling the movement and storage of cargo containers at its ports. With a vast network of terminals across six continents, DP World is recognized as one of the largest port operators globally.

The organization offers a comprehensive range of services related to port operations, including container handling, vessel berthing, cargo storage, and logistics solutions and many more. DP World's state-of-the-art facilities and advanced technologies ensure efficient and secure handling of cargo, promoting seamless trade flows. In addition to port operations, DP World also provides an array of value-added services to enhance the efficiency and effectiveness of the global supply chain. These services include customs clearance, freight forwarding, warehousing, and distribution, enabling end-to-end logistics solutions for customers.

DP World continually invests in innovation and digitalization to optimize port operations and provide smart logistics solutions. The organization leverages advanced technologies such as automation, artificial intelligence, and Internet of Things (IoT) to enhance operational efficiency, improve productivity, and enhance the overall customer experience. By operating in the business sector of port and terminal operations, it plays a vital role in facilitating global trade, connecting economies, and driving economic growth. The organization's commitment to innovation and its extensive global network position it as a key player in the logistics industry.

OVERVIEW OF THE ORGANIZATION

DP World's diverse range of products and services caters to the needs of businesses across various industries involved in international trade and logistics. By offering comprehensive solutions and leveraging technology-driven innovations, DP World aims to enhance supply chain efficiency and drive economic growth.

CARGOES is a DP World suite of enterprise services and products for the world of logistics and trade. Various products under CARGOES by DP World are:

1. **CARGOES Flow:** CARGOES Flow is a comprehensive logistics management system that enables end-to-end visibility and control over cargo flows. It streamlines the movement of goods, optimizing supply chain efficiency and providing real-time insights for effective decision-making.

2. **CARGOES Runner:** CARGOES Runner is a mobile application designed to facilitate efficient cargo operations on the go. It allows users to track and manage cargo shipments, access real-time information, and collaborate with stakeholders, empowering them to stay connected and informed throughout the logistics process.

3. **CARGOES PCS:** CARGOES PCS (Port Community System) is a digital platform that connects various stakeholders within a port community, including port authorities, customs, shipping lines, and logistics providers. It streamlines and automates port-related processes, such as vessel operations, cargo documentation, and customs clearance, fostering collaboration and improving efficiency.

4. **CARGOES Rostering System:** CARGOES Rostering System is a workforce management solution that optimizes rostering and scheduling of personnel in the cargo handling industry. It helps ensure efficient resource allocation, productivity, and compliance with labour regulations, enabling effective workforce planning and management.

5. **CARGOES PMT:** CARGOES PMT (Port Management and Tracking) is a system that provides real-time monitoring and tracking of cargo movements within a port environment. It enhances operational visibility, improves security, and facilitates efficient management of cargo handling operations in ports.

6. **CARGOES ZMS:** CARGOES ZMS (Zone Management System) is a solution that enables efficient management of cargo storage zones within a logistics facility or terminal. It provides tools for inventory control, allocation of storage space, and optimization of cargo handling processes, ensuring effective utilization of storage areas and enhancing operational efficiency.

7. **CARGOES Customs**: CARGOES Customs is a solution that simplifies customs processes and compliance for cargo shipments. It automates customs documentation, facilitates electronic data exchange, and ensures adherence to customs regulations, enabling smooth and efficient customs clearance for imported and exported goods.

INTERNSHIP EXPERIENCE

As a Summer Intern at DP World, I had the opportunity to be part of the software development team as a Full Stack Developer. During my internship, I actively contributed to the design, development, and implementation of software applications and solutions.

Working as a Full Stack Developer, I was involved in both frontend and backend development tasks. On the frontend side, I collaborated with the team to create user interfaces and experiences using responsive designs and utilized frameworks like Svelte and Tailwind CSS to develop modern and intuitive user interfaces. On the backend, I worked with frameworks like Node.js, Express, and Nest.js to build robust and scalable server-side applications. I developed RESTful APIs, integrated with databases like Prisma and PostgreSQL, and implemented business logic to support various functionalities. Throughout the internship, I actively participated in the entire software development lifecycle.

In addition to coding, I also gained experience in software testing and debugging. I worked closely with the quality assurance team to conduct unit tests, identify and resolve issues, and ensure the overall stability and reliability of the applications. Working in DP World's software development team provided me with valuable exposure to industry best practices, agile methodologies, and collaborative development processes. I had the opportunity to work on real-world projects, leveraging modern technologies and tools.

Overall, my experience as a Summer Intern as a Full Stack Developer in DP World's software development team allowed me to gain practical skills, broaden my understanding of software development practices, and contribute to the organization's digital transformation efforts.

CHALLENGES FACED

As a full stack intern, some of the biggest challenges I encountered were:

Learning Curve: The field of full stack development is vast and constantly evolving, which meant there was a steep learning curve. Understanding and becoming proficient in multiple technologies, frameworks, and languages required dedicated effort and continuous learning.

Collaboration and Communication: As a full stack intern, I had to collaborate with various team members, including designers, backend developers, and stakeholders. Effective communication, coordination, and ensuring alignment of goals and expectations across different roles and teams were essential for project success.

Keeping up with Technology: The technology landscape in full stack development is dynamic, with new frameworks, libraries, and best practices emerging regularly. Staying updated with the latest trends and incorporating new technologies into projects while ensuring compatibility and stability was a continuous challenge.

Despite these challenges, I embraced them as opportunities for growth and learning. Overcoming these hurdles allowed me to develop a stronger skill set, improve my problem-solving abilities, and gain valuable experience in the field of full stack development.

INTRODUCTION

Project Introduction

Insurance is a risk management tool that provides financial protection against potential losses or damages. It involves an agreement between an individual or a company (the insured) and an insurance company (the insurer). The insured pays a premium in exchange for the insurer's promise to compensate for covered losses or damages. In the context of logistics, insurance in logistic operations helps mitigate financial risks, protects against unexpected losses, and provides peace of mind to logistics operators and their customers. It is an essential component of a comprehensive risk management strategy in the logistics industry.

Cargoes insurance specifically refers to insurance coverage for goods or cargo being transported. It provides protection against risks such as loss, damage, theft, fire, or other perils that can occur during transit. Cargoes insurance ensures that the value of the goods is protected, giving peace of mind to both the sender and the recipient. It is a crucial component of logistics operations to safeguard the financial interests of all parties involved and to ensure the smooth and secure movement of goods.

CARGOES SHIELD, an ongoing project by DP World is a technology solution offering instant marine cargo insurance to the existing members of enhancing customer experience, streamlining insurance processes, creating convenience, efficiency, accessibility & solving underinsurance. It tightly integrates with other digital products of CARGOES by DP WORLD like CARGOES COMMUNITY and CARGOES LOGISTICS by instantly creating premiums for members of CARGOES COMMUNITY or LOGISTICS.

CARGOES SHIELD is going to become a market place for all commercial cargo insurance in the near future because of its streamlining processes for cargo insurance.

Motivation

The motivation to create a cargo insurance product stem from the recognition of the critical role that insurance plays in mitigating risks and protecting businesses involved in the logistics industry.

Goal of the project:

To provide comprehensive coverage and financial protection for goods and commodities involved in logistics operations. By offering insurance coverage, the goal is to protect businesses against potential losses resulting from theft, damage, loss, or unforeseen events during transit.

ACCOMPLISHMENTS
(PSEUDOCODES & ALGORITHMS)

BACK-END WITH TESTING

1. *Created end-points of a web application*

The typical flow of data in this architecture is as follows:

a. Client (e.g., web browser) sends a request →
b. Controller receives the request →
c. Controller extracts data from the request and performs validations →
d. Controller invokes the appropriate service →
e. Service processes the request, performs business logic, interacts with databases or external systems →
f. Service returns the result to the controller →
g. Controller formats the response, applies any necessary transformations or mappings →
h. Controller sends the response back to the client.

DTOs are often used to define the structure of the data exchanged between the client and the server or between different layers of the application, helping in enforcing data contracts and providing a clear representation of the data being transferred.

2. *Method to retrieve data from Redis Cache*

```
1. Attempt to retrieve the data from the Redis cache using the get method.
2. If the data doesn't exist:
        a. Make a call to the external service to retrieve the data using the various methods of the appropriate client.
        b. Set the token in the Redis cache using the set method with an expiration time of 1 hour (60 * 60 seconds).
3. Return the data.
```

3. *API calls were made and handled*

```
1. Log an informational message indicating the start of the process, with relevant details such as the operation being performed.
2. Prepare the headers for the request, including the necessary authentication credentials or API keys.
3. Send a request to the appropriate API endpoint, using the base URL and endpoint specific to the desired operation.
4. Await the response from the API, if the response indicates that the request failed, handle the error appropriately.
5. Return the response.
```

4. *Added interaction handlers and DTOs to check for requests and responses(50+ DTOs were extended)*

1. Define the interaction handlers as classes or functions that handle specific interactions between the client and the server.

2. Identify the different types of interactions that need to be handled (e.g., handling user registration, processing payment, fetching user data).

3. For each interaction handler:
   - Define the necessary input parameters required for the interaction.
   - Implement the logic to handle the interaction, which may include:
     - Validating the input data.
     - Performing necessary operations, such as querying a database, making external API calls, or invoking other services.
     - Applying business logic or rules.
     - Generating appropriate responses or triggering events based on the result of the interaction.

4. Ensure that the interaction handlers are organized and reusable, promoting code modularity and separation of concerns and test them individually to ensure they function correctly and produce the expected results.

5. Integrate the interaction handlers with the appropriate routing or controller mechanism of your web framework to handle incoming requests.

6. Handle any errors or exceptions that may occur during the interaction and provide meaningful error messages or responses to the client.

5. *Created new database tables to stage and seed data (Around 5 tables were added)*

1. Define the schema for the new database tables in the Prisma schema file (prisma/schema.prisma)
2. Use the Prisma schema language to define the table structure, fields, relationships, and any constraints.
3. Run the Prisma migration command to generate the necessary SQL migration scripts
4. Use the Prisma CLI to generate the migration files based on the changes made in the Prisma schema.
5. Run the command npx prisma migrate dev or yarn prisma migrate dev to create the migration files.
6. Apply the migrations to the PostgreSQL database:
7. Run the Prisma migration command npx prisma migrate dev to execute the migration files and create the new database tables.
8. Prisma will generate and execute the necessary SQL statements.
9. Create seed data files with the necessary data in a suitable format (e.g., JSON).
10. Write a script or use Prisma's seeding feature to load the seed data into the respective tables.
11. Execute Prisma CLI command npx prisma db seed to populate the tables with the seed data.

Maintain the schema and data integrity by applying further migrations and seeds as needed during the development lifecycle.

6. *Integration testing with API calls*

1. Define test scenarios covering different API functionalities.
2. Set up a test environment resembling production & prepare test data for various API use cases.
3. Develop test cases with HTTP requests, headers, payloads, and expected responses.
4. Configure testing frameworks (e.g.,) or tools (e.g., Postman).
5. Execute integration tests by sending requests and capturing responses.
6. Validate responses against expected criteria (status codes, payloads).
7. Capture and report test results, including failures.

7. *Used in-built and custom decorator validations*

1. Identify fields requiring validation in data models or DTOs.
2. For built-in decorator validation (e.g., using class-validator library) for decorators like @IsNotEmpty(), @IsString(), @IsNumber(), etc., to validate the data type and presence of values.
3. For custom decorator validation create custom decorators by defining validation logic using the decorator syntax and functions.

```
export class CustomValidation implements ValidatorConstraintInterface
{
  validate(value: any, args: ValidationArguments) {
    // Custom validation logic
    // Return true if value is valid, false otherwise
  }
  defaultMessage(args: ValidationArguments) {
    // Return error message for validation failure
  }
}
```

4. Use decorated models or DTOs in controllers or services.
5. Validate input data using decorator validation.

8. *Component testing done using PactumJS (Around 100+ test cases were written)*

Pseudocode:

```
// Set up Pactum test suite
describe('Component Testing with Pactum', () => {
  let api;
  // Set up before hook to initialize the API instance
  before(() => {
    api = pactum();
  });
// Test scenario 1
  it('should return a successful response', async () => {
    // Define the expected request and response
    const expectedRequest = {
      method: 'GET',
      path: '/api/users/1',
    };
    const expectedResponse = {
      status: 200,
      body: {
// body of the response payload
      },
    };
```

9. *Performance Testing done using k6*

Pseudocode:

```
export default function () {
  // Send a GET request
  const res = http.get('http://localhost:3000/api/users');
  // Validate the response
  if (res.status === 200) {
    console.log('Request successful');
  } else {
    console.log(`Request failed with status code: ${res.status}`);
  }
  sleep(2);
}
```

10. *Added migrations & seeds to the database and extracted information from them*

Pseudocode:

```
where_clause(filters: GetQueryDto): DatabaseQuery {
  let whereClause: DatabaseQuery = {};

  if (filters.one) {
    whereClause.one = filters.one;
  }
  if (filters.two) {
    whereClause.two = filters.two;
  }
  return whereClause;
}

select_clause(): DatabaseFields {
  return {
    one: true,
    two: true,
  };
}
```

FRONT-END WITH TESTING

1. *Implemented policy summary page*

Using Svelte and Tailwind, pages developed based on a user interface design created in Figma.

2. *Implemented additional information page*

Pages developed using Svelte and Tailwind CSS, following a design from Figma. Implemented responsive layouts and touch-friendly elements for optimal mobile experience.

3. *Enhanced the frontend to a mobile friendly application*

1. Tailwind CSS's responsive utility classes were employed to adapt the user interface design seamlessly across various screen sizes.
    a. Responsive layouts created using Tailwind CSS's grid system.
    b. Custom styles applied using Svelte's media queries or Tailwind CSS's responsive classes.
    c. Consideration given to touch-friendly design for buttons, links, and input fields.

2. Thorough testing was conducted on actual mobile devices and browser dev tools' mobile emulation to ensure a smooth user experience across various devices.

4. *Extended custom components for user interaction*

1. Define the Component: Create a new Svelte file with markup, styles, and data properties.
2. Import Dependencies: Bring in any external dependencies or libraries if needed.
3. Export the Component: Make the custom component accessible to other parts of the application.
4. Use the Component: Incorporate the custom component in the Svelte markup of another file.
5. Pass Props: Send data to the custom component using HTML-like attributes.
6. Handle Events: Define event listeners or emitters within the component to handle user interactions.
7. Style and Customize: Apply Tailwind CSS or inline styles to customize the component's appearance.

5. *Added custom rules to validate inputs*

```
Pseudocode:

function isRule() {
  return {
    validate: (value, name) => {
      if (!<regex>.test(value)) {
        return `<error-message> `;
      } else {
        return ''; // Empty string indicates validation passed
      }
    },
  };
```

6. *Added validations on sanctioned countries, commodity mappings and user balance*

```
Pseudocode:

function _get(url, message, options) {
  try {
    enableLoader(true, options);
    const response = await fetch(url, {
      method: 'GET',
      headers: getRequestHeaders(),
    });
    const responseJson = await response.json();
    if (response.ok) {
      return responseJson;
    } else {
      options.message = message;
      showToasterNotification(options);
      handleUnauthorizedAccess(response);
      throw new ResourceError(message, responseJson);
    }
  } finally {
    enableLoader(false, options);
  }
}}
```

NOTE: Created several functions and pages which cannot be included in the report due to project confidentiality.

USER FLOW

1. When a user visits the landing page, they are presented with an intuitive interface that captures their attention and guides them through the insurance process.
2. Through the use of advanced algorithms and real-time data, an instant premium is calculated and displayed to the user, providing them with an immediate estimate of the insurance cost based on the information they provide.
3. To gather more detailed information about the user's specific insurance needs, an application form is presented. This form collects relevant data such as personal details, cargo specifications, and coverage requirements, ensuring that the insurance product can be tailored to the user's unique circumstances.
4. Once the user completes the application form, the system processes the information and generates a certificate that serves as proof of insurance coverage. This certificate includes vital details such as policy number, effective dates, and coverage limits, providing the user with a tangible document that confirms their insurance purchase.
5. The generated certificate can be instantly accessed and downloaded by the user, providing them with immediate access to their insurance documentation for record-keeping or further dissemination to relevant parties, such as shipping agents or regulatory authorities.

TECHNICAL OVERVIEW

TECHNOLOGY USED	ICONS	TECH
PROGRAMMING LANGUAGES		JavaScript
		TypeScript
		NodeJS
FRAMEWORKS AND LIBRARIES		NestJS
		PactumJS
		K6
		Svelte
		SvelteKit
		Tailwind CSS
		Playwright

DATABASE MANAGEMENT SYSTEMS		Prisma
		PostgreSQL
		Redis
API DEVELOPMENT AND INTEGRATION AND CI/CD		Kubernetes
		Azure
		Docker

ARCHITECTURE OVERVIEW

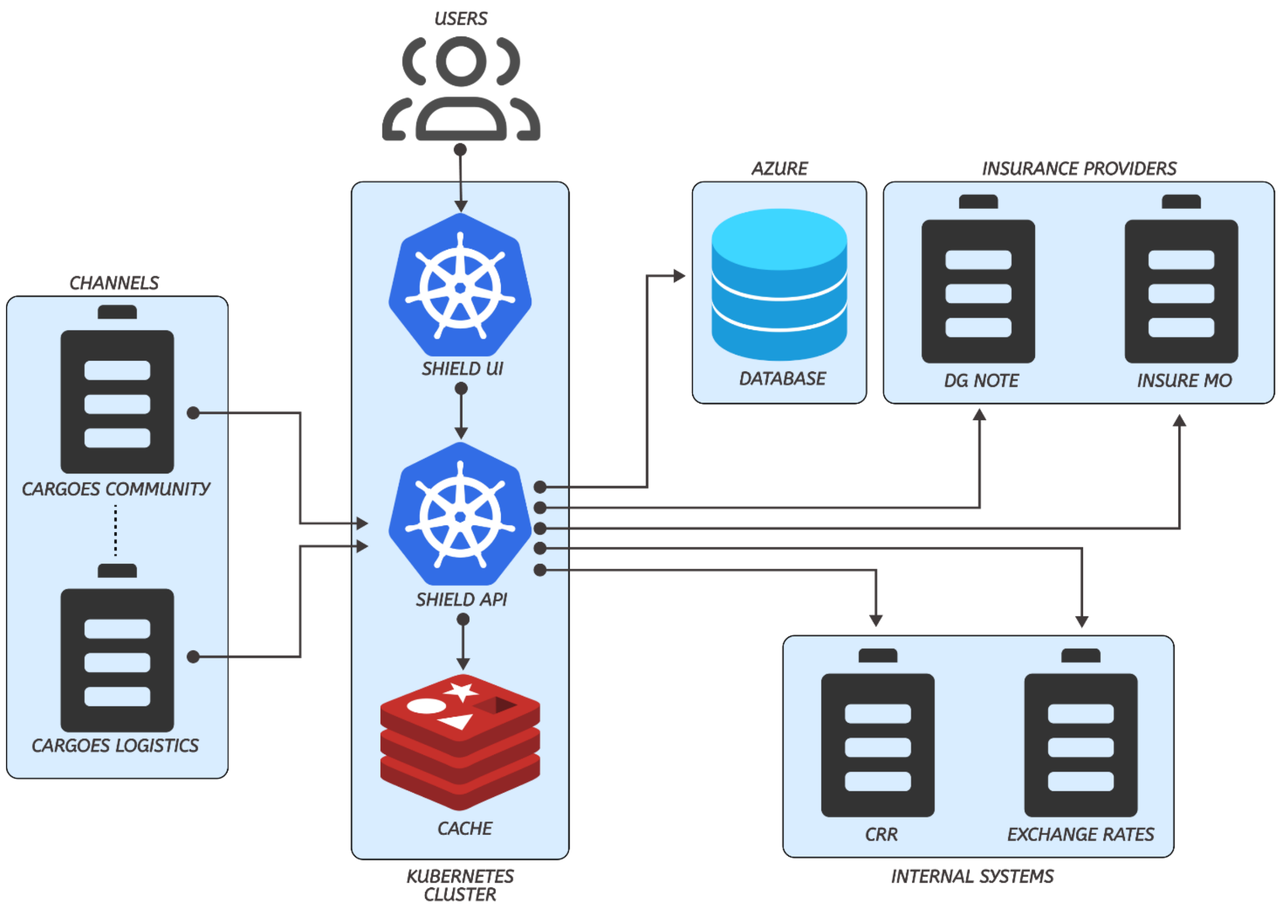

Fig.1 High Level Architecture

The flow of data and interactions in the architecture can be summarized as follows:

- Users interact with the Shield UI, accessing various features and functionalities.
- The UI sends requests to the Shield API, which processes the requests and handles business logic.
- The business process involves obtaining the booking number from DGNote (the insurer) and sending a Quotation Request to the application. Users are then prompted to provide their specific details in the application form in order to receive an instant premium.
- The Shield API communicates with Azure for database operations, such as storing and retrieving data, interacting with insurance providers, and integrating with internal systems. The Shield API may utilize the cache component to store and retrieve frequently accessed or computed data, enhancing performance.
- The Kubernetes cluster ensures the scalability, fault tolerance, and efficient deployment of the Shield UI, Shield API, and cache components.

Frontend (SHIELD UI)

The frontend of the Cargoes Shield Application is developed using the following technologies:

- Svelte : Svelte is a radical new approach to building user interfaces. Whereas traditional frameworks like React and Vue do the bulk of their work in the browser, Svelte shifts that work into a compile step that happens when you build your app. Instead of using techniques like virtual DOM diffing, Svelte writes code that surgically updates the DOM when the state of your app changes.
- SvelteKit : A framework for building web applications with Svelte, providing routing and server-side rendering capabilities.
- Tailwind CSS : A utility-first CSS framework for designing responsive and customizable user interfaces.

Frontend Testing

- Playwright : A powerful testing framework that can be used to automate end-to-end testing of web applications. It supports multiple browsers, including Chrome, Firefox, and Safari.

Backend (SHIELD API)

The backend of the Cargoes Shield Application is developed using the following technologies:

- NestJS : A progressive Node.js framework for building efficient and scalable server-side applications. Under the hood, it makes use of robust HTTP Server frameworks like ExpressJS . It provides an out-of-the-box application architecture which allows developers and teams to create highly testable, scalable, loosely coupled, and easily maintainable applications. The architecture is heavily inspired by Angular.
- Prisma : An Object-Relational Mapping (ORM) tool that simplifies database access and management. It unlocks a new level of developer experience when working with databases thanks to its intuitive data model, automated migrations, type-safety & auto-completion.
- Postgres : A powerful, open-source relational database management system used for persistent data storage.
- Redis : An in-memory data structure store used for caching frequently accessed data.

The backend handles the core business logic, data storage, and retrieval. It provides a RESTful API for the frontend to interact with.

Backend Testing

- PactumJS : A flexible and easy-to-use testing library for API testing. It allows developers to write declarative and expressive API tests using a fluent API.
- k6 : A popular open-source load testing tool that can be utilized to assess the performance and scalability of the backend infrastructure. It allows developers to write load test scripts in JavaScript, simulating multiple virtual users concurrently accessing the system.

GOALS & AMBITIONS

After completing my summer internship as a Full Stack Developer, my ambitions and goals in the field of Full Stack development have been further solidified. I have gained valuable experience and exposure to various aspects of the development process, which has ignited my passion and enthusiasm to pursue a career as a Full Stack Developer.

My ambitions include honing my skills and deepening my knowledge in both front-end and back-end development technologies. I aim to become proficient in popular frameworks, expanding my expertise in both client-side and server-side development. Additionally, I aspire to improve my understanding of databases, cloud computing, and DevOps practices to enhance my ability to design and deploy robust and scalable applications.

One of my goals is to become proficient in creating responsive and user-friendly interfaces. I aim to master UI/UX principles and apply them effectively to create intuitive and visually appealing web applications. Another important goal for me is to improve my problem-solving and debugging skills. I want to become adept at identifying and resolving issues efficiently, as well as writing clean and maintainable code that adheres to industry best practices and coding standards. Furthermore, I am keen on expanding my knowledge of software development methodologies such as Agile and Scrum, and collaborating effectively with cross-functional teams to deliver high-quality software solutions.

Overall, my ambitions and goals as a Full Stack Developer after completing my summer internship revolve around continuous learning, skill enhancement, and staying up-to-date with the latest technologies and industry trends. I am determined to contribute to impactful and innovative projects, and ultimately, establish myself as a competent and well-rounded Full Stack Developer in the industry.

CONCLUSION

In conclusion, the internship provided valuable insights and practical experience in the field of software development. Throughout the internship, various tasks were undertaken, including developing web applications, creating UI pages, implementing interaction handlers, and performing database operations. The internship also involved testing and validating the applications through functional testing, performance testing, and authentication testing. The experience gained during the internship helped in enhancing technical skills, understanding industry practices, and working collaboratively within a professional team. Overall, the internship provided a comprehensive learning experience and contributed to personal and professional growth in the field of software development.