DC3 - Blockchain Dashboard

by

Alon Cohn - 123456
Anubhav Guha - 123456
Kiran Mainali - 406393
Pradyumna Krishna Kashyap - 123456
M Haseeb Asif - 406219

A Project submitted to

Technische Universität Berlin
School IV - Electrical Engineering and Computer Science
Department of Telecommunication Systems
IOS

Project documentation

July 24, 2019

Supervised by:
Marcel Müller

Eidestattliche Erklärung / Statutory Declaration

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

I/we hereby declare that I/we have created this work completely on my/our own and used no other sources or tools than the ones listed.

Berlin, July 24, 2019 _____

Berlin, July 24, 2019 _____

Berlin, July 24, 2019 _____

Berlin, July 24, 2019 _____

Berlin, July 24, 2019 _____

Acknowledgement

We would like to express our deepest appreciation to all those who have helped us to successfully complete this project. A special gratitude to our supervisor Mr.Marcel Müller, whose contribution in terms of guidance, advice and helping us to keep motivated throughout the project.

We would like to thank the SNET for providing us this opportunity to work on the project with a set of enthusiastic team members and a great mentor. We would also like to thank all other mentors, friends, motivators, critiques, supporters and well-wishers who have helped us finalizing this project within the limited time frame.

Thanks,

Group 6

Abstract

The purpose of this documentation is to present our work on the Internet of services lab project. The project requirement was to develop a simple dashboard for web parcel management. To establish that we created an interface system that could be used by courier-companies\customers\postmans to monitor and update the status and condition of the package. The package data is then stored on the blockchain to obtain ease of access to the data and consensus among the stakeholders, which implies that the data would be cross-company owned. To obtain the data from the blockchain to our dashboard we have also created a back-end communication level API. This system also enables all parties involved in the package delivery to communicate with a single non-centralized source.

Contents

1 Introduction

1.1 Background

As companies and consumers increasingly purchase goods online, the demand for cross carrier management platform delivery services grows (First Research 2013, 7). Furthermore, the growth of online retail sales has influenced the logistics industry for the past ten years and the trend is expected to continue at least on a similar level during the next few years. (Delfmann et al. 2002, 203.)

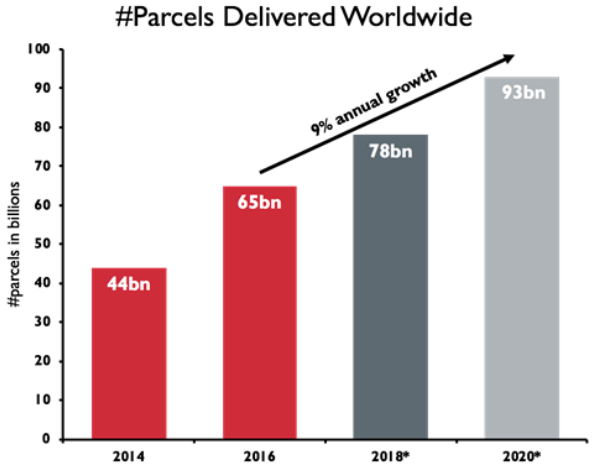


Figure 1.1: Parcels Delivered worldwide [billions]

Responding to the increased demand of small-sized frequent shipments incurred by e-commerce has become one of the biggest challenges for logistics express delivery companies. A successful delivery of shipments to consumers distributed across large geographical areas will require re-designing of the existing system.

The increase of business-to-consumer (B2C) e-commerce activities implies that business are border less, global e-commerce is selling products or offer services across the world.

The idea to investigate delivery service based on the blockchain proposed by the TU Berlin. The DC3 team in this project did not focus on implementing the blockchain system but rather in creating a platform interface to manage the parcel. At the end of the day, DC3 aims to be a part of a B2B blockchain solution that will make cross-border logistics much more reliable and efficient than they are now.

Parcels are not always delivered at their destination in time and with the right agreed quality, this can cause financial losses due to reimbursements and low customer satisfaction. Especially

higher value shipments ranging from routine parcels up to extraordinary parcels are insured against risks of loss, theft, damage or any other events that could impact delivery precision and quality. Prompt and secure detection, monitoring and recording of the shipment events is needed to allow tracking of service quality level, accountability, liability evidence in disputes and for analysis and optimization of the logistical chain.

DC3 (Dashboard for Control and Communication center) is a web-based dashboard for monitoring and controlling the logistics transactions beyond the border of a certain logistics company while leveraging DiLLaS (distributed ledger for logistics and supply chain management (DiLLaS).

DiLLaS is an IoT and Blockchain solution that offers a new and unique view on shipment events data for logistics companies and their partners. These views create a detailed insight and transparency of significant security and safety events during the shipment. Additionally, by storing them in the distributed ledger, a trusted and decentralized recollection of the events of the shipment is created. DC3 enables a new way of handling parcel delivery and accelerates the implementation of more reliable and financial sustainable delivery processes.

The goal of DiLLaS is to provide a distributed ledger for the secure and trustworthy storage of events that occur during the transport of goods. A DiLLaS Mobile App will act as a client for the ledger and is intended as a means for all participants of a logistic chain to securely log a registration, deregistration and handover of a parcel within the distributed ledger. In combination with GRAVITY, the DiLLaS Mobile App will in addition be able to check the status of sensors that are attached to the parcels such temperature or humidity sensors and to log violations of predefined delivery conditions within the distributed ledger. Since DiLLaS makes use of Blockchain technology to persist the events and their related data in an immutable, transparent and a distributed manner where common incidents such as lost or delayed parcels can be securely traced back to its originators. With the commonly approved log of logistic events in DiLLaS, logistic operators will be able to improve their efficiency in terms of parcel delivery time by identifying bottle necks, costs per parcel by identifying unreliable partners and the quality of service by evaluating if parcels are delivered according to the predefined delivery conditions such as a constant temperature or an acceptable intensity of shocks for goods of high value.

1.2 Problem Discussion

1.2.1 Vendors communication and limitations

There are multiple inefficiencies in cross-border delivery industry, related to lack of transparency and trust between the different logistics providers. International parcel delivery usually go through several logistics vendor, each on their side of the border, and there is a lot of accounting and interoperability overhead in the interactions between the different vendors. The idea is to solve these issues with the blockchain technology, taking advantage of its inherent transparency and trust. From a Logistic company perspective, the parcel information is limited to their ownership and post completion of handover to a peer logistic company the package would not be efficiently tracked and the interrelated companies handling the package would not be in consensus. This common interface provides a plausible solution to the problem stated above by giving an opportunity for the companies to keep track of the package

irrespective of the ownership of the package at a given point of time by having full information about it. It will also help the companies achieve better customer satisfaction by providing such sensor options to its customers to make use of.

1.2.2 Vendors HMI

From a customer perspective, when a customer orders a parcel the information revealed to him is depended on the quality of the carrier web interface, with a unique high-detailed dashboard the customer can be independent from the retail interface system and be informed about the status of the package and location of the package at all times throughout the journey of the package from source to destination.More options are made available to the customer in terms of a temperature and a shock sensor which is capable of taking lower and upper limits from the customer as an input and reverting with any irregularities in the package.

2 Related Work

Historically messages were usually hand-delivered using a variety of methodologies, the most common being runners and horseback riders. Then came the introduction of mechanized courier services which differed from ordinary mail services by features such as speed, security,tracking and individualization of express services. They not only started providing such services within a town or a city but started offering worldwide services. The problem being very evident with such services is tracking these packages as stated above. Such offer services are made worldwide, typically with hub and spoke model. The spoke-hub distribution paradigm is a form of transport topology optimization in which the companies organize routes as a series of "spokes" that connect the outlying points to a central "hub"[1].

These companies further utilize courier software which provide electronic proof of delivery and electronic tracking details of the package limited to ownership. Currently the gap incorporated among logistic companies on cross country deliveries are handled using basic communication or with the help of some third party companies. Eurosender is one such company that provides a platform for businesses and individuals to have a unified solution to run their logistics processes globally. They provide services for customers ranging from a simple package delivery to Freight transport. They also help their customers track the package based on individual tracking numbers produced by each of the logistic companies for the delivery of the packages.

Further with respect to look and feel of the application being developed, we were provided with a front-end SPA template "Crystal React Bootstrap Dashboard". Crystal React Bootstrap is a multipurpose admin dashboard created using the technologies React,Redux and Bootstrap. Its main goal is to help create complex applications using a lot of simple and easy to use React components that are embedded in this template. This template is helpful to create many kinds of dashboards relating to the health sector, online social networking websites and also for companies in need of a dashboard.

This template mainly consists of the below mentioned simple packages that could be useful for the development of an end-to-end application.

- Charts
- Buttons
- Notifications
- Sweet Alert
- Redux Form
- AirBnB React Dates
- Google Map and Uber Vector Map

- React Bootstrap Table
- React Big Calendar

3 Concept and Design

DC3 dashboard has 3 different types of personas - Company, Customer and Postman. Hence we had to design the system differently for each user persona. Our front end application is a Single Page Application (SPA) using react JS. Our back-end is developed by using Node JS leveraging Express framework.

3.1 Architecture

DC3 follows a modular approach for the whole system. As shown in the bigger picture, entry point for the system is social login, Google in this case. we are using passportJS which can be used to integrate other social strategies e.g. GitHub, Facebook as well. Furthermore, passport does provide local strategy as well where we can configure local database as well for authentication and authorization. This feature would be useful when we different vendors does deploy their own instances of the diLLas service and want to authorize the existing customer base from their own system. we have detailed overview for google authorization explained later as well

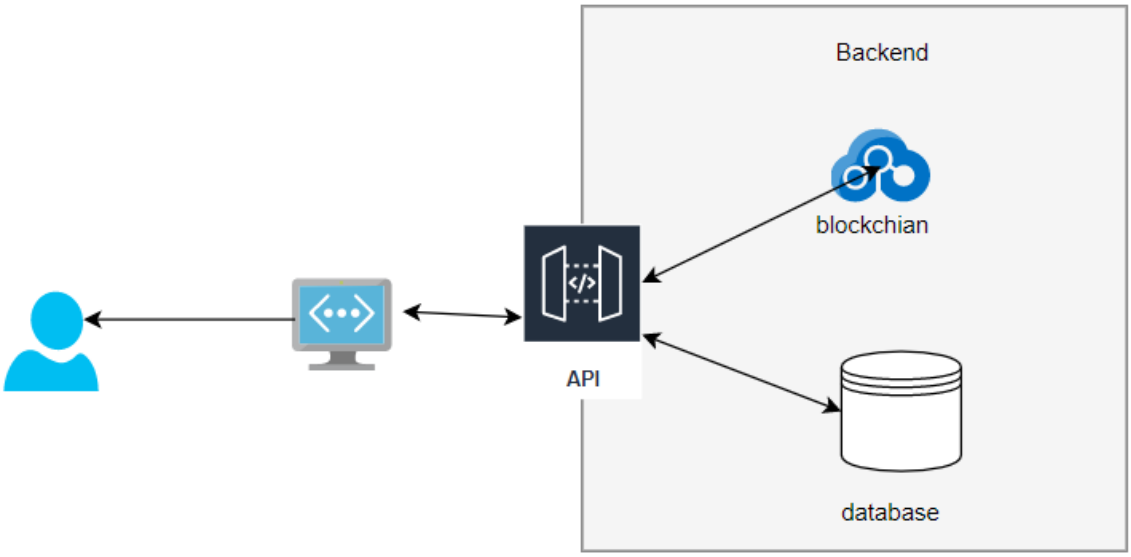


Figure 3.1: System Architecture(abstract view)

Once user is logged-in, for example from browser, an access-token is stored into the local session storage for being used on the front end SPA (Single Page Application) react app. This token is used for further communication between front-end and back-end. It contains user

profile and authorization info. This info is leveraged to protect the different endpoints on the server side and provide related data based on the user type. For example, when a customer hits the /packages endpoint he will be able to see his own packages only but when a company user hits the same endpoint, he will get all the company level packages in the response.

3.2 Authentication and Authorization with Google

Authentication is a vital part of the project and most real world applications need authentication and authorization. While authentication identifies some entity as a valid user, authorization defines the actions that the user is allowed to perform, based on his/her roles and rights. There are many solutions to provide for authentication or authorization in an application and we had two major choices,

- First one was to keep user state management on Redux.
- Second was to use a Facebook or a Google Authentication integration with our application.

We chose to go with the second choice as it was a new learning for the team to integrate it with our application, reduces the burden for the users to register themselves rather directly sign in with their Google accounts and more importantly because it provides back-end services to securely authenticate users, paired with easy-to-use client SDKs. It can authenticate users using passwords and federated identity provider credentials.

In our application, once the user clicks on the Google authenticate button,it directs the user to Google's OAuth 2.0 server, which requests access to the user's meta data from the Google drive with a read only scope. Further after granting or denying the access to the user, is then redirected to the original page, which pareses the access token from the string. The page then uses this access token to make a simple API request which calls the Drive API's to retrieve the information about the authorized user's Google account. If this request is successful then the API response is logged in the browser. For the first time user's we check our local database to determine the existence and add the user to our database if the user name does not exist and fill in the vital information using the above fetched information from the authorized user's Google account. If the user name exists before hand, we validate the token from the server side with the OAuth2.0 provider and once validated, a JSON Web Token (JWT) is produced to securely provide authorization to the front end. We use a main component in our database table "Persons" that is "PersonType" component which determines the role given to a user.Based on this value we determine if the user is either a customer,postman or a company representative. The role with the lowest features is the role of a user, which is the value provided by default to the user on first time log in and henceforth the company role has authorization to upgrade the user to a postman or a company representative. Initial company role setup is done by the admin. The below figure provides a overview of the functioning of the Google authentication and Authorization based on role integrated to our DC3 application.

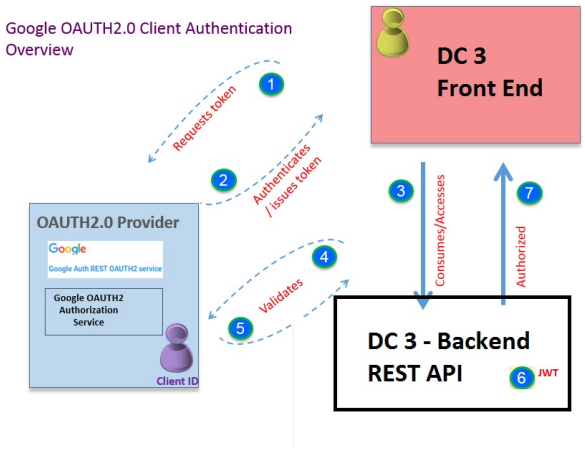


Figure 3.2: Authentication & Authorization

3.3 Database Design

DC3 database has 8 tables. Order, Person, Order History and Order sensors are the major tables while others play an important role as well. Designs majorly circulate around Orders table. We have normalized the database to avoid storing redundant data e.g. Order and Sensors had many to many relationship so we divided it and created another table, OrderSensors. Let's understand the philosophy behind each table

3.3.1 Company

The company table is persisting the information about all the companies. It stores the company name and a brief description of the company.

Id	Int - auto increment	Unique company ID
Name	varchar	Company Name
Description	varchar	Brief company description

Table 3.1: Company table

3.3.2 Address

Address table is maintaining all kind of address across the whole system, be it order adree or customer home address. Hence it has relationship with Order and Person. Order table has two address Ids, each for pick and delivery address of the order

3.3.3 Sensor

The sensor table is a repository of unique sensors we have available in the system. It will store the names, and different possible thresholds a specific sensors has. It was designed in a generic way to store the different sensors in the same table.

AddressId	Int - Autoincrement	
StreetAddress	varchar	
City	varchar	
Country	varchar	
PostCode	int	

Table 3.2: Address table

Id	Int - Autoincrement	
Name	varchar	
MinValue	varchar	Minimum possible reading for the sensor
MaxValue	varchar	Maximum possible reading for the sensor
DisplayUnit	varchar	

Table 3.3: Sensor table

3.3.4 Person

The Person table is storing various kinds of user information. It stores personal information for the company, customer and postman. Additionally, it maintains the social login values for the user. Once a user signs up, it defaults to a customer but admin or company user can change his role to postman or admin (company) user.

Id	Int - auto increment	Unique company ID
FullName	varchar	
Email	varchar	
Password	varchar	Minimum possible reading for the sensor
DateOfBirth	varchar	Maximum possible reading for the sensor
PersonType	varchar	Person type e.g. customer, postman or company
PersonRole	int	Stores person role or associated company id
GoogleProviderId	varchar	Google user-id
GoogleAccessToken	varchar	Logged-in user access token

Table 3.4: Person table

3.3.5 Orders

Orders table is keeping most of the data in the database and it will have a higher churn rate than any other table in the whole system. It stores a lot of referential ids from other tables than actual data e.g. pick & drop address, person, company.

Id	Int - auto increment	Unique Order ID
OrderID	Int - AutoIncrement	
PickAddressID	Int	Pick up address id
DropAddresID	Int	Drop address Id
PickDate	date	Registration or pick up date
ArrivalDate	date	Delivery date
PersonID	int	Package Sender id
ReceiverPersonID	int	Package Receiver id
Status	varchar	E.g. Registered, In-Transit or Delivered
CompanyId	int	Reference for associated company

Table 3.5: Orders table

Id	Int - auto increment	Unique company ID
Name	varchar	Company Name
Description	varchar	Brief company description

Table 3.6: OrderSensors table

Id	Int - auto increment	Unique company ID
Name	varchar	Company Name
Description	varchar	Brief company description

Table 3.7: Incident table

3.3.6 OrderSensors

3.3.7 Incident

3.3.8 Database Diagram

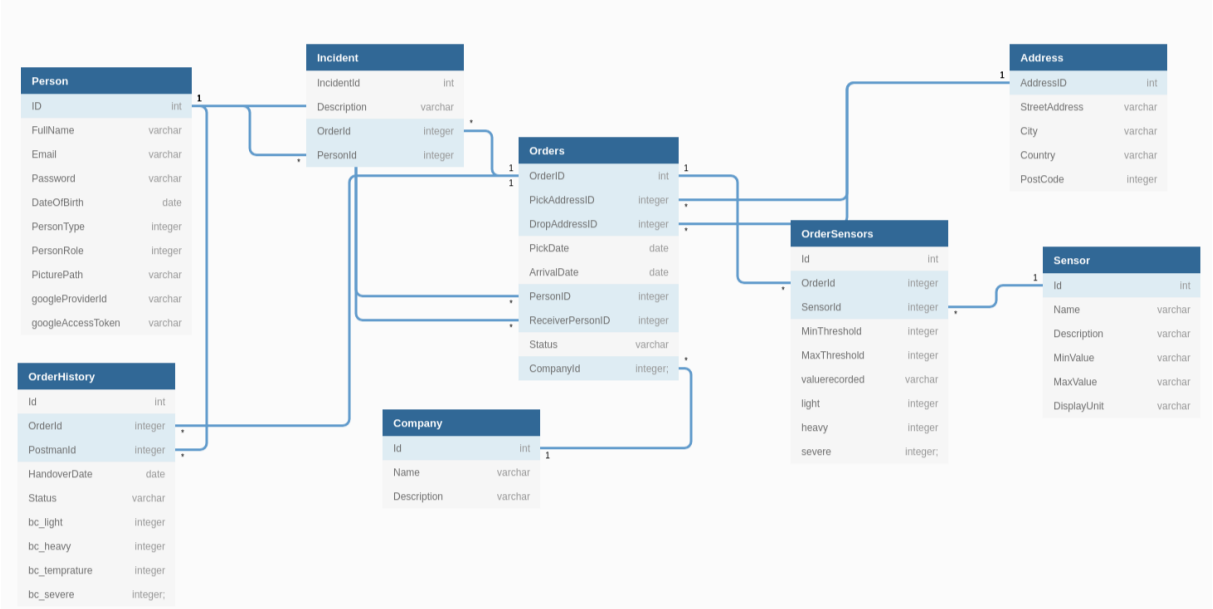


Figure 3.3: Database Diagram

4 Project Organization

This chapter will cover how we organized the entire project. We will uncover the the project timeline in the first section. How roles are distributed among the members will be mentioned in second section and we will talk about tools used in the third section

4.1 Timeline

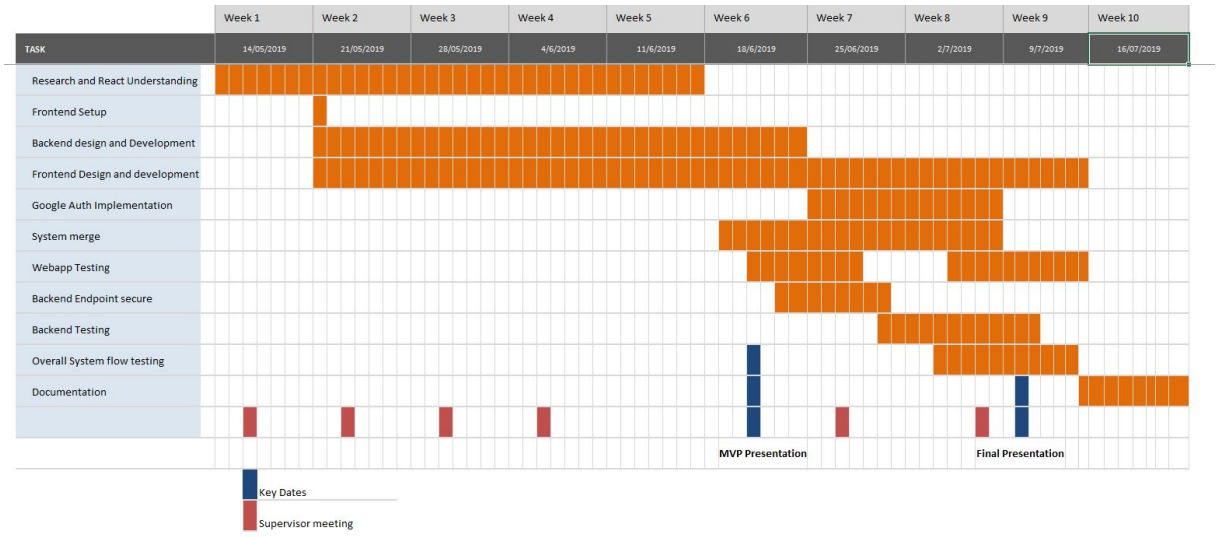


Figure 4.1: Project Timeline

4.2 Work Distribution

Task	**Team Member**
Customer	Alon
Company	Kiran
Postman	Anubhav
Integration, Routing and Common Web pages	Pradyumna
Backend	Haseeb Asif
Documentation	All Members

Table 4.1: Task Distribution

To fulfill the requirement of the project in the given 10 weeks time frame, we divided the task as shown in **table 4.1**. Member who took charge of each user space design worked from the same basic design and were synchronized with each other to maintain the consistency. Github was used by all members fro source control purpose. Member responsible for each user space had the responsibility to develop the functionality of those user too. Member who was working on customer view was responsible for all customer functionality and so on for other team members. Integration, routing and common web page development task handler was responsible for login/register page, overall page routing, system integration with backend. Member with backend task was responsible for overall database design and implementation according to the need system requirement. Documentation is done by all member input. The table shows the major responsible person for the given task, however each member has provided input on other's jobs too. Project was collective task using input, knowledge and expertise of each and every member of the group.

4.3 Tools and Infrastructure

- Github - Source control (iosl-dc3, iosl-backend)
- Trello -Task mgmt (https://trello.com/b/RXwlgcJI/iosl-dc3)
- Slack - Communication (https://iosltu2019.slack.com)
- Overleaf- Documentation Collaboration (https://overleaf.com)
- VS Code - Code editor
- Azure for Postgres SQL - Database

5 Implementation

In the chapter we explain how the implementation is done for both the frontend and the back-end sides. We first elaborate on the different roles exist in the frontend:

5.1 Frond-end

IDE: Visual Studio code
Language: React
Libraries used:

- React-router-dom
- Receiver mail must be a register user

5.1.1 Home Page

http://localhost:3000/#/

After a successful login all roles (postmen, company, customer) is being redirect to the same home page. the home page present a table which contains a minimize version of the package data. the user can here get a full image regarding the amount of packages and the correspond status.
Algorithm:

1. Extracting the user role. Each user as part of the google authentication has a role type. The roles is number 1-3 which mention the user role (postman, customer, company)
2. **http://localhost:8000/packages/user/** get request, before component mount: upload package n information. The information will be display as a table.

5.1.2 Costumer

Libraries used:

- rc-slider for creating the temperature slide bar
- react-horizontal-timeline for the time line
- redux-form for the registration form

The costumer role refer to any individual who send a package through the system. The costumer has the following abilities:

- Register a package

- Delete a package
- Show a package detail view
- Time line

5.1.2.1 Register a package

http://localhost:3000/#/packages/registerPackage

The register package process divided by two component:

- RegisterPackage.js - view
- UserSpace.js - controller

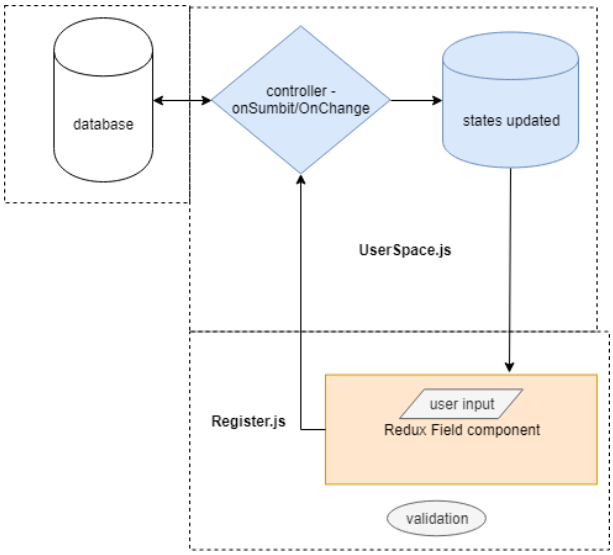


Figure 5.1: register package process, component diagram

The component ResgiterPackage is an assembly of Redux <Field> component and React input. for each input field the user enter, the page is rendered and the state is updated The form contain the following validation:

- No text input filed can be left empty
- Receiver mail must be a register user

i.e without standing in the form requirement, the user can not press the submit button.

When registering a package, a sensor data may be provided by clicking the correspond checkbox. It is import to mention that not every package has a sensors data.
Algorithm:

1. **http://localhost:8000/address** post request,enter the address and get back and ID for each address provided
    a) **http://localhost:8000/packages** post request, enter the packages to the packages tables, return a package ID - address ID should be provided

i. **http://localhost:8000/OrderSensors** post request, enter the sensors (if exist) to the sensors table, return a sensor ID - package ID should be provided

b) **http://localhost:8000/OrderHistory** post request, enter the action(registration) as part of the order history for later on use in the time-line - package and sensor id should be provided

Only after fetching the data to the database occurred successfully the page content will be replace to a "package have being registered" message.

5.1.2.2 Cancel package

http://localhost:3000/#/packages/active

The feature cancel package allows the user to cancel a package. After pressing the navbar "delete package" option, the user will be directed to the Activs.js compnent.
The user will be shown a data-minimize table only with packages in a "register" status, **any other package which has being dispatch already can not be cancel**.
From the system perspective while the package is being canceled, the data is not deleted from storage,it will only be marked as canceled.
Algorithm:

1. **http://localhost:8000/packages/user/** get request, before component mount: upload all packages with "register" status
2. If user press delete:
   a) **http://localhost:8000/packages/:id** put request. Change the spesific package to "cancel" mode.
   b) **http://localhost:8000/OrderHistory** post reqest. Update the timeline, add the event to history

5.1.2.3 Detailed package view

http://localhost:3000/#/package/ID

The feature allows the user to view the extensive package information and the package timeline. After pressing the specific package ID in the home page,the user will be redirected to rout.The ID will be transfer as a prop from the home page to the the Detailed.js component. In addition to the package data which was given during the registration time, the table present the current status of the package and the condition of the package.i.e if the user ask a for a sensor the table will present the amount of events where the condition were not full fill. The sensor data will be only present if the user register sensors to the package.
Algorithm:

1. **http://localhost:8000/packages/details/id** get request, before component mount: upload package n information. The information will be display as a table

2. **http://localhost:8000/packages/orderHistory/id** get request, before component mount: upload package history.the history will be sent to the Timeline.js component as a prop.
   a) Timeline.js use the npm package "react-horizontal-timeline" to display the package timeline. the time line present the status of the package the company and postman name which holds the package at the moment.

5.1.3 Company

Company user in the system will be able to do the following activities:-

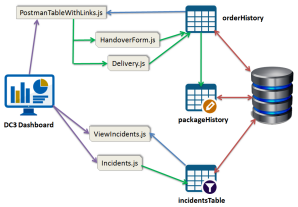
- **View package pickup request sent by customer to the particular company.** When customer registers package, package is registered to DC3 database with registered status. Company will see assigned package on the system filtered by status.
- **Accept package pickup request from customer my assigning to the postman under the company.** The registered package is accepted by company by assigning to available postman of the company for pickup from the customer location. Company does this by clicking Assign button which calls assign.js. Company has to enter the email address of postman on the form and click the button. After the package is assigned to particular postman the status of package will be change to Transit in the database. There is Assign Package Sub-menu under User Management for the package assign function
- **Upgrade users to postman or another company user of the same company.** Company can upgrade registered user to postman or to another company user of the same company by entering email address. For the upgrade function to be applied to particular user. The user has to register themselves as customer first. There is sub menu on sidebar under User Management for this function. UpgradeUser.js is used for this functionality.
- **View tracking history of packages under the company.** When company user login to the system, in the dashboard there will be all the list of packages that have been under the particular company. By clicking on order ID on the leftmost side it takes to detail page of particular company with tracking history of the packages shown as timeline.
- **View user list of own company and change their role to available roles in the system.** Company can see user list of and upgrade individual to postman or another company user as discussed above.
- **Create incident for the packages** Company can create incident for the particular packages if any alteration on the condition is noticed. For this purpose a separate menu is established on sidebar as Incident Management. Incident are created by putting Package ID and incident description under Create Incident sub-menu.
- **View and resolve the incident of particular packages.** Company can view and resolve the incident for packages if those incident has been resolved. For this company has to click resolve incident button on the incident list displayed in Incidents sub-menu under Incident Management menu.

5.1.4 Postman

Overview of the functionality:

Postman user in the system will be able to do the following activities:-

- **View all the packages that have been assigned to him:** (PostmanTableWithLinks.js) When the Company user assigns a package (PackageId) to a particular postman (PostmanId) then a relevant record is created into the table orderHistory. All the packages that are assigned to the current user are fetched using http://localhost:8000/orderHistory
- **View the information about the package such as pickup address, drop address, status etc:** The table orderHistory contains columns that hold the relevant data of the package required for the postman to identify every package.
- **Able to change the status of the package as Delivered once the actual delivery has been made:** whenever a postman physically delivers the package to the recipient at the respective Drop Address, then he is able to press the button 'Deliver' that invokes the function to setthe status of the Package as 'Delivered' and enter a new row in the packageHistory table only to denote that the package is now counted as one of the delivered packages.
- **Able to Handover the package by assigning it to another Postman and change the status accordingly:** whenever a postman reaches an intermediate point in the route of the package where he has to hand it over to another postman, then the 'Handover' button is used to redirect to another component HandoverForm.js where the user id of the receiving postman needs to be entered. This functionality allows one postman to assign a package to nother postman by changing the 'AssignedTo' data in the relevant table. After this subroutine is invoked the handed over parcel starts to appear in the dashboard of the receiving postman.
- **Able to raise Incident whenever there is any violation of policies:** A postman to who the package has been assigned or handed over is able to judge the condition of the package and raise an incident mentioning the details of the package and the nature of violation also mentioning its current condition using the form created in the Incident.js component.
- **View the Incidents that have been raised in the past:** Under the heading 'Incident Management' there is a link 'View Incidents' that redirects to ViewIncidents.js component that pulls data from the Incidents table and allows the Postman to see the relevant list of Incidents raised by or against him that have not yet been resolved.



Postman.png Postman.png

Figure 5.2: DC3 Postman

5.2 Back-end

Back-end or server is a layer on top of the database and Blockchain for the front-end. It does handle the authentication and authorization and does take care of fetching and sending data to the hyper-ledger fabric. We are using following technologies

IDE: Visual Studio code
Language: JavaScript
Frameworks: NodeJS with Express

5.2.1 Project Structure

Project structure has is modular and functionality has been divided into folder based on functionality. Hence we have two major folders at the root, server and test, with some other application level files

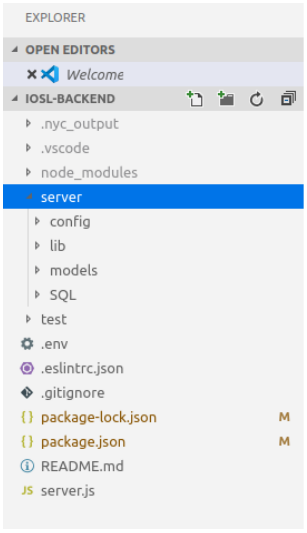


Figure 5.3: Directory Structure

- **Server**: This folder contains all the logic related to server functionality.
  - **Config**: This folder contains the configuration for the server side. authkeys.js does contain the secret keys used across the application for google social login and to generate the token for authorization. Passport.js contains the logic for google strategy. we can write another strategy in the same file as well e.g. github or local strategy. In future, When we are going to integreate the platform with different vendors, we will be using the local passport strategy using the vendor database. After that, route.js does contain all the routes for the application. any new routes needs to be registered in this file. Finally token.util.js contains utility functions to generate and send the token to the client back. It also exposes the method to generate the token which is used for mocked unit testing.
  - **lib**: Library or lib folder does contain the middleware for our back-end application. Right now we have only one, secureMiddleware, to secure our API endpoints. It does check if access-token header is provided in rquest body, cookie or querystring and then decode it using our secret. If successful it will set the user profile on the request object, otherwise throw 401 response.
  - **models**: Models folder does contain all the different models to fetch the data from database or block chain. Each file represent an entity in the system or a related

database relation. Furthermore, index.js file contains the references of all the models and is used almost everywhere in the system as single reference point for the models.

- **SQL**: SQL folder contains the database schema which can be used to deploy or create the database while setting up a project on a new machine. Additionally, it contains a schema file to generate the database diagram on dbdiagram.io as well.

- **test**: This folder contains all the unit tests. Each file contains the tests for the respective model file. Test does use mocha and chai frameworks.
- **env**: .env is environment file to setup variables at the start of the project instead of reading from the config file, it will be read from the environment. This file is git ignored. It isn't used at the moment as we are reading db config from the db.js directly for now.
- **eslintrc.json**: ESlint configuration can be changed using this file. We are using Airbnb base syntax for our linting.
- **gitignore**: GitIgnore is used to tell git which files we don't want to store in the source control.
- **package.json**: This file contains all the configuration to setup the project. When we do npm install, it will read this file and install all the package on end user machine. It also contains the dev dependencies. We have different scripts configured to run e.g. running the server or run server in the debug mode, run linter.
- **server.js**: This is the starting file where our project starts. This file will bootstrap the application by configuring different modules, followed by loading the routes and then running the http server on specific ports. We also configure the express server, body parser (which parse the request body), cookie-parser, passport js and most importantly CORS to send cross origin requests.

5.2.2 Unit Testing

We are using mocha framework for unit and integration testing in our project. Additionally we have used chai as an assertion framework for our tests. Each test has a title or test case description followed by request and expected results. A sample test is as follows

```
/*
* /GET company/id
* Test the single company records from the database
*/
it('It should GET single company', (done) => {
  chai.request(server)
    .get('/company/2')
    .end((err, res) => {
      res.should.have.status(200);
      res.body.should.be.a('array');
      res.body.length.should.be.eql(1);
      done();
    });
});
```

We used chai request to call the endpoint, set the access token for the secured endpoints, and then compare the results with the expectation. We also have tests which will send different access token to validate the authorization of the endpoints e.g. postman can see only packages assigned to him and not all the packages or a customer cannot request a handover for a package.

6 Evaluation

DC3 web app visualizes logistic process from blockchain data and IOT sensors monitors. DC3 system puts customers in loop with regard to package status merging and displaying relevant information from blockchain, DC3 database and IOT sensors. From registering the package to delivery at final destination customer does not have to look for different vendors and tracking options. Fully informed package condition to the exact custody holder of package is being managed by using the DC3 system.

6.1 System Flow Evaluation

For the evaluation of developed system we created separate email (gmail account) for each user. Created two for companies users, two for customers and four for postman. Postman were divided into two companies for delivery process. We started from adding users to system and registering the package to delivering the registered package through the companies postman.

As there is no specific procedure on system to add user as company directly, first companies were added as customers and later changed their roles to company. And similarly postman and customer were added in system using google authentication. Also postman are initially registered to system as customer which later are upgraded to postman by company users. And company can upgrade users to another company users for same company.

For the flow of delivery of package and how that information will be displayed on system we started with the registration of package from the customer filling all the details required. Customer has to assign listed company as their initial package handler. The receiver has to be registered in the DC3 system too. Customer can choose available sensors from the system that should be attached in the package to check the condition and safety of package (which is optional). For now DC3 has two sensors type to be selected Heat Sensor and Drop Sensor. User can select either of the two sensors and or neither of the sensors.Customer can un-register the package until it is assigned to the postman by the company.

Customer registered package are displayed to the company account, where company can assign package to the postman of the company for pickup and handover process. For the package to be able to assign to the postman by company package should have registered status. Company will assign package to postman by entering the registered email address of the postman.

Postman sees the list of available job under the account and postman can deliver package to the receiver, handover package to another postman of same company or handover package to another postman of different company. Postman has to enter the email address of receiver and and click the delivery button for delivery of package. And for handover purpose postman will have to enter email address of other designated postman. In between the delivery process incident creation is the function where users(company, postman and customers) can create

the incident regarding the package issues. Customer can report to the company if package has arrived distorted, broken or in unacceptable condition. Postman can generate report if the handed package is broken, torn, distorted or in unacceptable condition and company can generate incident if they notice unusual condition on packages under their custody. The raised incident are displayed to postman, company and customer who holds the package.

Resolve incident provides mechanism to solve issues and resolve created incident in the individual user level. Company, postman and even user can resolve created incident if they are satisfied with the action taken by the concerned parties to mitigate the raised issues.

Each package detail page shows the detail information of the package with the timeline. Timeline helps to visualize the package status and where is the package cat the current time.

6.2 System Limitation and Recommendation

DC3 webapp has covered all stakeholders (customer, Postal Companies and Postman) within the system and the basic flow of system has included all these stakeholders inside the package delivery loop. But there are some limitations in system which are as below.

- By using google authentication, system does not directly detects user type. Like if company wants to register in the system first they have to register as customers and have to manually upgrade user to the company. And same condition applies for postman too.
- DC3 currently does not have a mechanism to fetch data directly from blockchain. Which means IOT triggered incident is not working in the system. Only user created incident stored in system database are displayed in the Incident system.
- DC3 webapp is only tested in local environment, in local PCs and laptops. No real time testing and stress testing is done for real business scenario.
- Coding standard is not up to the mark as most of us project members were beginners in react JS and many of the functionalities and layout were reused/modified from the existing given template. So there might me issue of some redundant and unused files, although we have cleaned our repository leaving only necessary files.

Even with these limitations DC3 provides the basic functionality that were presented to us during initial project kickoff meeting and several supervisor meetings. The system is basic product which can be expanded by adding features in the future like fetching Blockchain information, directly adding different users in system from register/login page and IOT triggered sensor incident creation.

7 Conclusion

In the beginning we were given a brief idea about the project explaining that it is a dashboard for monitoring logistics data. As per the requirements it was clear that the web application is supposed to be used by three different user groups. Hence the primary focus was on the look and feel of the landing screen, navigation pane and dashboard components. Development in this part involved plenty of hit and trials; adding and removing components; re-designing old components, changing and adjusting size and placement of components on the dashboard therefore making sure that each user group has a different view of the dashboard. Using reach JavaScript makes it very convenient to embrace the hit and trial approach of work because we are able to view the changes immediately into the running local server, once the code changes are saved in VS Code. The presence of NavBar component provides a rigid classification of different components; for example Register.js is kept under Package Management category and Incident.js is kept under Incident Management category. This ensures the whole webpage does not get refreshed while navigating from one functional component to another. Considering the diversity among our team members in knowledge background and areas of expertise, it was not an easy job to immediately start working on the project development. We needed few weeks to introduce ourselves to React JS and once we started developing the web pages, we learned a lot about this technology while fiddling with the individual functionalities of every component. This knowledge and experience we have gained while developing DC3 dashboard is very priceless and some of us even look forward to pursue this track of full stack development with React/Node JavaScript and make it an integral part of our careers in Information Technology sector. We have also come across several challenges during the development phase that we have overcome with teamwork, brainstorming, collective research and individual hardwork. It was a brilliant experience to work as a team with these incredibly skilled members especially Mohammed Haseeb Asif, who single-handedly overlooked the entire development phase serving as a highly capable Project Manager with his years of impeccable experience on Software Industry.

List of Tables

List of Figures

Appendices