Connect

powered by Cognizant

Elizabeth Coady

Software Developer Level 4

Table of Contents

Introduction

My name is Lizzie and I work for the digital consulting company Cognizant. The company offer IT solutions to clients who want to digitize their business. This can include services like moving their servers to the cloud, remaking their software and being the middle man in connecting clients & external start-up companies. People who join as software developers in the company will be deployed to a client site or an internal team to develop projects. My first project I started on was for a financial client in a pod of 9, with 1 more on shore pod and 4 off shore pods, totalling around 60 people. On a big team of people, I was able to learn and pick up a lot knowledge in a short space of time, and began being able to independently start working on tasks, collaborating with the team and making decisions that would impact the project. I have talked about this project in the document 'External Cognizant Project Portfolio Submission'. After a few months in July, I moved to an internal company project with a much smaller team of 6 people. The team wanted to on board a developer to develop an MVP of a project concept. I worked with 2 members of the team to start from the beginning of creating user stories, paper prototypes of the UI and as I was the only developer, give a technical perspective of the possibilities of the features we could include. I would then independently create the software design and begin creating the application from scratch. I used a Trello board to track my progress which my 2 team members who I was working with would also use to make adjustments where needed when we would do regular reviews of the progress, adhering to an Agile way of working. An exported pdf version of this board can be found as the attachment called 'MVP Task Flow _ Trello'. We decided to use an Agile approach as it was a company normality, I had had experience on previous projects using an Agile approach which worked well and we wanted the team to be able to review regularly and give feedback as the app was mainly to be catered towards them, essentially making the team user testers. This in turn did cause a delay in our initial MVP launch date as we added more backlog from feedback, but it created a better catered user product overall so it was the better decision to use an Agile way of working rather than Waterfall.

Project Overview

My project is a platform to view and search 'solutions'. Solutions are, for example, a piece of software developed by another company that you may want to use in your own project. My platform has many different solutions stored on a database and are then able to be swiped through on a mobile UI and searched on. The team have decided to build this app as they 1; want to have a central database to store all the solutions and track them as an admin system since there is no repository to currently store this, more directed in the style of a website and 2; present these solutions to client facing employees, and eventually clients so they can look at potential solutions they would want to get involved with, this side being additionally in the form of a mobile app to swipe through. The initial solutions presented to them will be newly added and recommended to them based on their field & interests. This gives the finding of solutions to work with more engaging for client facing people and will be able to create a partnership between them quicker and easier. For these ideas, we decided the name 'CONNECT' for the platform would be best suited.

Stakeholders

My team consisted of 7 people when I joined with a further 2 people joining 5 months later. The project consisted of 2 tracks, the physical track and the platform track. The physical track was the team actively engaging collaboration with clients and scale ups, the platform track was the team involved in the app development, myself only being involved in the platform track.

The team members were:

Joris Bouwens – Product Owner (both tracks); created wireframes, user stories, conducted user testing and managed the project

Lisa Tendler – Business Analyst (both tracks); created wireframes, user stories

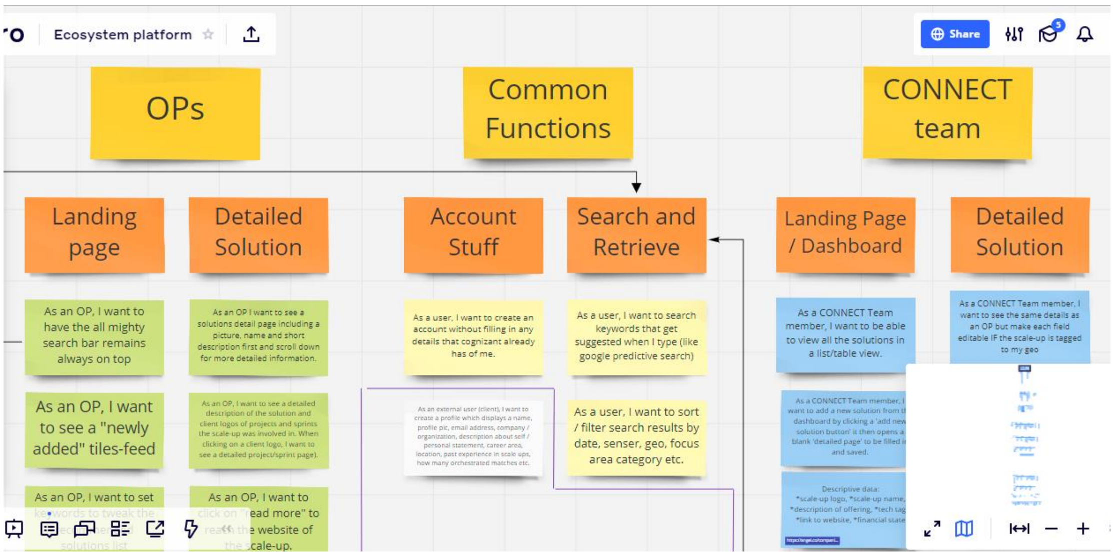
Alexander Stoeckl – Developer (platform track); joined at later stage to develop front end

Diogo Caraca – Developer (platform track); joined at later stage to develop front end

Natalie Blok, Henrik Engstroem, Jasper Nijdam, Mischa Seitz – Business Consultants (physical track); used and tested the app

Concept, Design and User Story creation

I, Joris and Lisa together wrote the user stories for the MVP and designed the user interfaces using the online tool Miro (Figure 1). I therefore had a say in what features were to be included or excluded due to technical limitations, time etc., which was vital as neither Joris nor Lisa had a technical background. I was able to see things from a business stand point & user experience, which was the main point that we felt would determine the success of the platform. For example, I decided that as the users of the app were primarily going to be internal employees of the company, a single sign on would be needed (Figure 2) as to prevent a user from registering and signing in. This demonstrates a response of the user journey we wanted to fulfil: convenience and ease. After this, Joris and I created a UML diagram show in Figure 4. The circles show which class types we wanted with the different attributes attached to each one, along with different relationship types to connect them all, an example of this being the relation from a Sprint type to Project type called 'resulted_in', meaning a sprint succeeded and moved into a full project.



OPs

Common Functions

CONNECT team

Landing page

- As an OP, I want to have the all mighty search bar remains always on top
- As an OP, I want to see a "newly added" tiles-feed
- As an OP, I want to set keywords to tweak the recommended solutions list

Detailed Solution

- As an OP I want to see a solutions detail page including a picture, name and short description first and scroll down for more detailed information.
- As an OP, I want to see a detailed description of the solution and client logos of projects and sprints the scale-up was involved in. When clicking on a client logo, I want to see a detailed project/sprint page).
- As an OP, I want to click on "read more" to reach the website of the scale-up.

Account Stuff

- As a user, I want to create an account without filling in any details that cognizant already has of me.
- As an external user (client), I want to create a profile which displays a name, profile pic, email address, company / organization, description about self / personal statement, career area, location, past experience in scale ups, how many orchestrated matches etc.

Search and Retrieve

- As a user, I want to search keywords that get suggested when I type (like google predictive search)
- As a user, I want to sort / filter search results by date, senser, geo, focus area category etc.

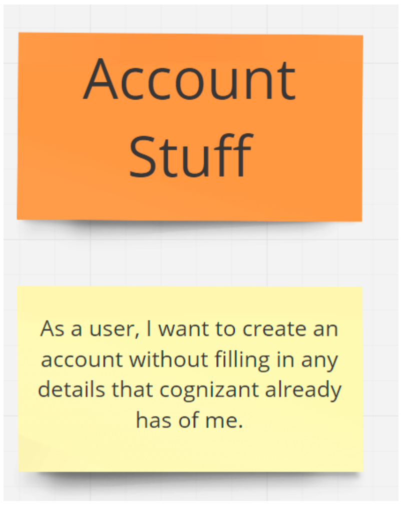
Landing Page / Dashboard

- As a CONNECT Team member, I want to be able to view all the solutions in a list/table view.
- As a CONNECT Team member, I want to add a new solution from th dashboard by clicking a 'add new solution button' it then opens a blank 'detailed page' to be filled i and saved.
- Descriptive data: *scale-up logo, *scale-up name, *description of offering, *tech tag *link to website, *financial state https://angel.co/compani...

Detailed Solution

- As a CONNECT Team member, I want to see the same details as an OP but make each field editable IF the scale-up is tagged to my geo

Figure 1 – Part of user story board made using Miro



Account Stuff

As a user, I want to create an account without filling in any details that cognizant already has of me.

Figure 2 - Close up of user story board

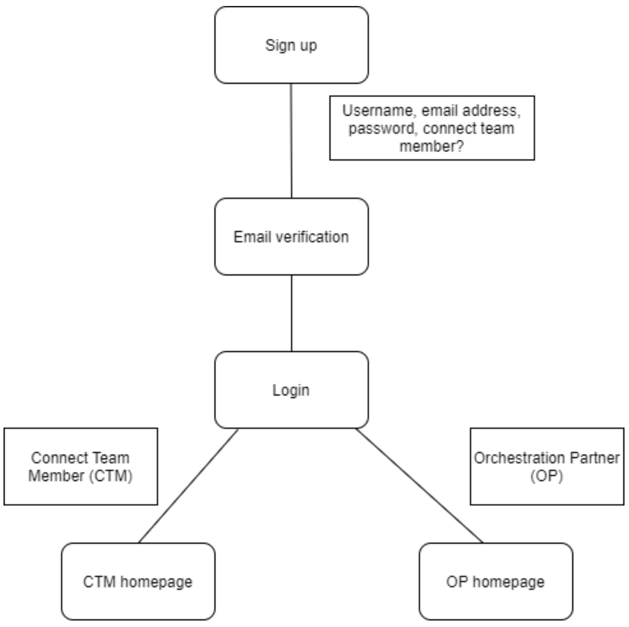

Sign up
Username, email address, password, connect team member?
Email verification
Login
Connect Team Member (CTM)
Orchestration Partner (OP)
CTM homepage
OP homepage

Figure 3 - Activity diagram of user sign up

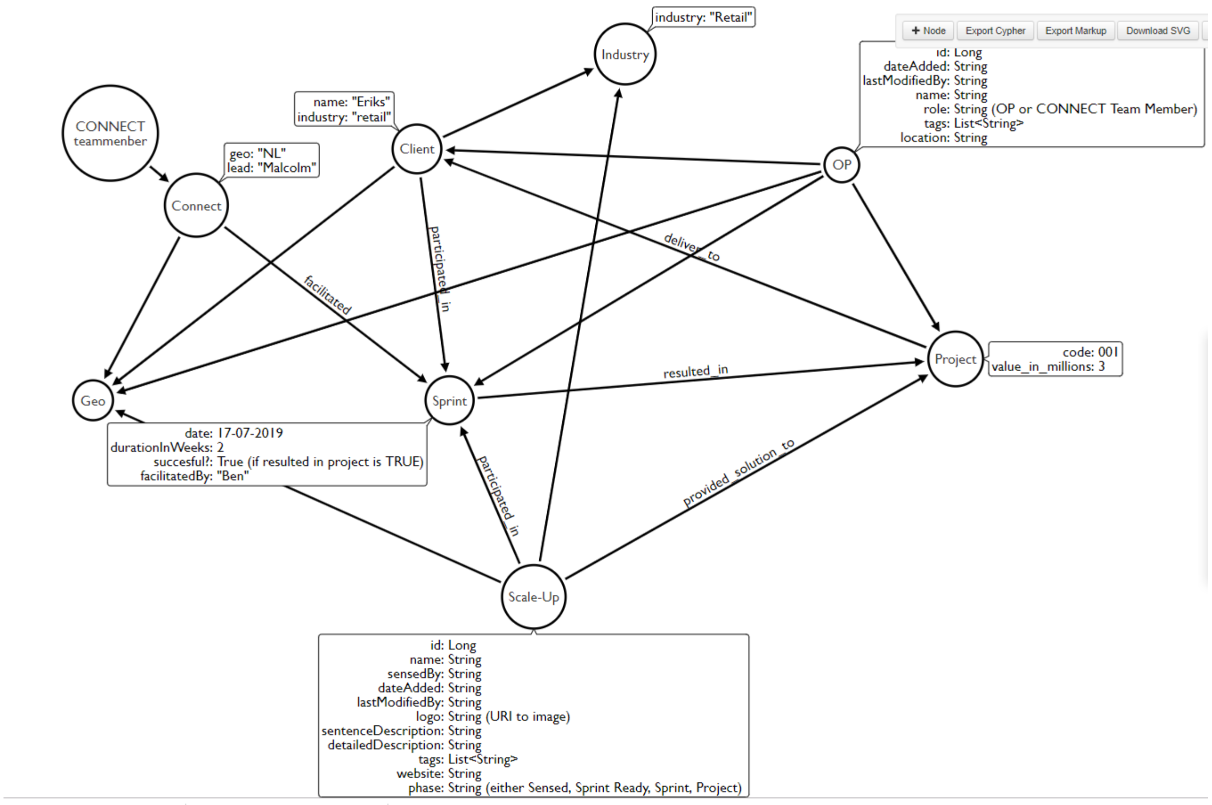


Figure 4 - UML diagram for the initial model & node structure

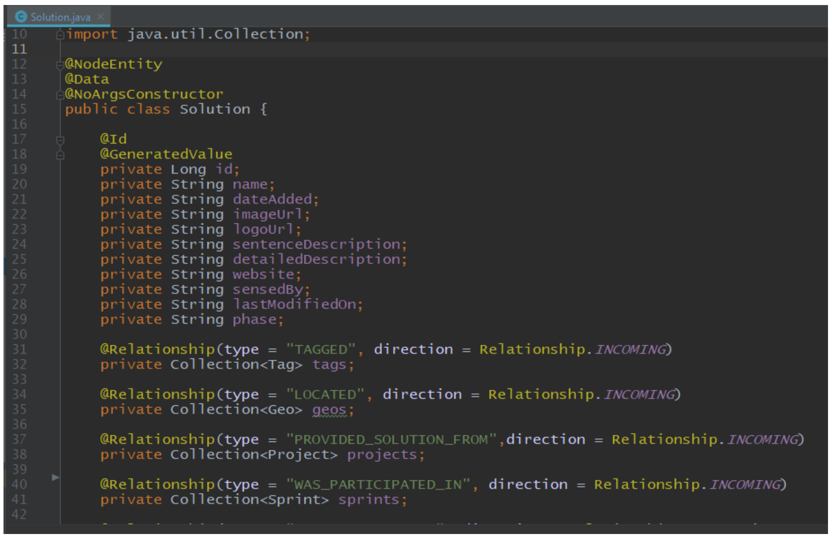
Tools used

When it came time to start building the app, I needed to decide on what tech stack to use. As this was going to be a web and mobile app, I decided on Java with Spring Boot as a backend, having used it before and being the company standard language. I chose the frontend framework Ionic with Angular as this was framework designed to work as both the web and mobile user interface, meaning there would be no need to create two separate codebases. I lastly chose Neo4j as a database. We wanted to use a graph database as it fit in right with the principal of the app, to create relationships between clients and solutions. For cloud services I used AWS, having experience using it before. This demonstrates using my own initiative as a developer to make decisions that will affect the project by choosing the final tech stack, and also demonstrates being able to correspond to business environments and working in a team as we jointly chose a graph database to satisfy the need of 'connecting' and the relationships between data being the central part of the app.

I used the IDE IntelliJ to code the Java + Spring backend and Visual Studio Code to code the Ionic Angular frontend. I used these IDE's as it makes the coding developing easier and more efficient, it provides mark-up, structure, code completion and debugging programmes, to list a few. I used different IDE's for frontend and backend as they are catered for and provide more and better support for the respective languages I am using for both.

Development

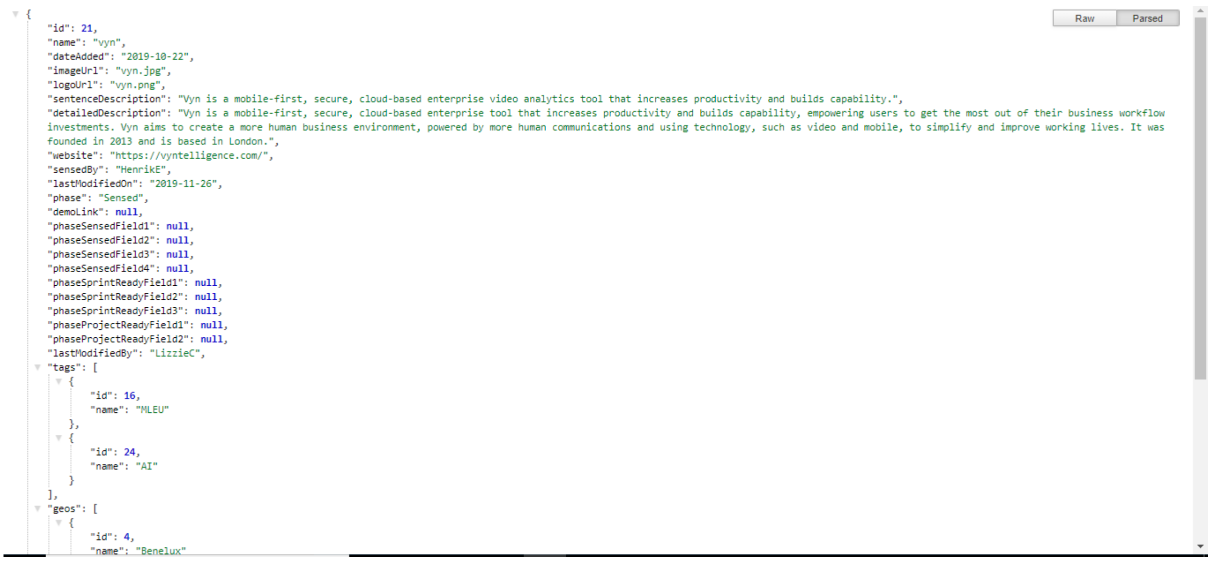
Backend



```java
import java.util.Collection;

@NodeEntity
@Data
@NoArgsConstructor
public class Solution {

    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private String dateAdded;
    private String imageUrl;
    private String logoUrl;
    private String sentenceDescription;
    private String detailedDescription;
    private String website;
    private String sensedBy;
    private String lastModifiedOn;
    private String phase;

    @Relationship(type = "TAGGED", direction = Relationship.INCOMING)
    private Collection<Tag> tags;

    @Relationship(type = "LOCATED", direction = Relationship.INCOMING)
    private Collection<Geo> geos;

    @Relationship(type = "PROVIDED_SOLUTION_FROM",direction = Relationship.INCOMING)
    private Collection<Project> projects;

    @Relationship(type = "WAS_PARTICIPATED_IN", direction = Relationship.INCOMING)
    private Collection<Sprint> sprints;

```

Figure 5 - Solution Model

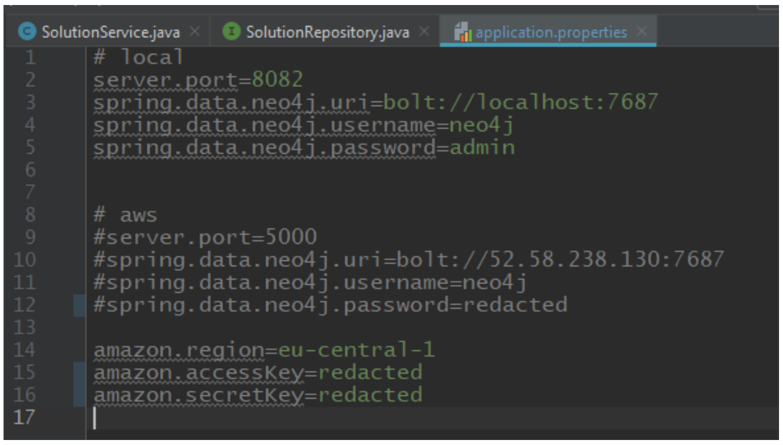
Starting with the backend, I used the practice of creating separate data models using the single responsibility principle, starting with 'Solution' and 'User' (Figure 5). I will show an example of a Java backend using Spring Boot using the Solution model as an example, I used dependency injection (repository -> service -> controller) to create the APIs I need.



```
{
    "id": 21,
    "name": "vyn",
    "dateAdded": "2019-10-22",
    "imageUrl": "vyn.jpg",
    "logoUrl": "vyn.png",
    "sentenceDescription": "Vyn is a mobile-first, secure, cloud-based enterprise video analytics tool that increases productivity and builds capability.",
    "detailedDescription": "Vyn is a mobile-first, secure, cloud-based enterprise tool that increases productivity and builds capability, empowering users to get the most out of their business workflow
    investments. Vyn aims to create a more human business environment, powered by more human communications and using technology, such as video and mobile, to simplify and improve working lives. It was
    founded in 2013 and is based in London.",
    "website": "https://vyntelligence.com/",
    "sensedBy": "HenrikE",
    "lastModifiedOn": "2019-11-26",
    "phase": "Sensed",
    "demoLink": null,
    "phaseSensedField1": null,
    "phaseSensedField2": null,
    "phaseSensedField3": null,
    "phaseSensedField4": null,
    "phaseSprintReadyField1": null,
    "phaseSprintReadyField2": null,
    "phaseSprintReadyField3": null,
    "phaseProjectReadyField1": null,
    "phaseProjectReadyField2": null,
    "lastModifiedBy": "LizzieC",
    "tags": [
        {
            "id": 16,
            "name": "MLEU"
        },
        {
            "id": 24,
            "name": "AI"
        }
    ],
    "geos": [
        {
            "id": 4,
            "name": "Benelux"
```

Figure 6 - JSON response of a get request

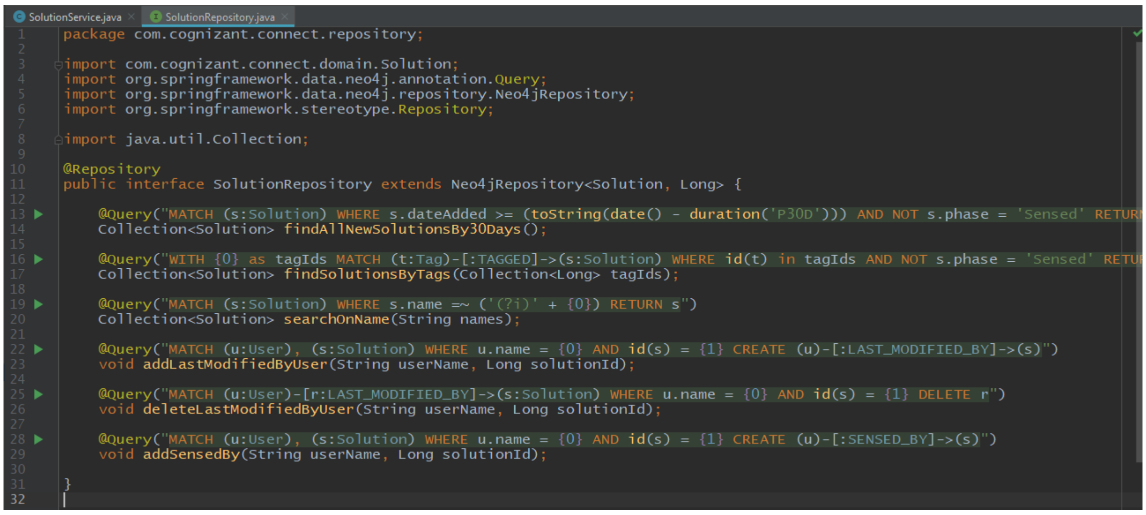
As I am creating a web API, the requests and responses will be in the JSON format. Figure 6 displays a JSON response of the model in Figure 5. The request to get this response is an API call to get a Solution object by id, using id as a path variable. The request sent to retrieve the response in Figure 6 would be 'http://localhost:8082/solutions/23', show in the function at the bottom of Figure 10. This is then able to be used by the frontend and interpolated to HTML, as seen in Figure 15. This demonstrates the use of data structures needed in the web developing environment.



```
1   # local
2   server.port=8082
3   spring.data.neo4j.uri=bolt://localhost:7687
4   spring.data.neo4j.username=neo4j
5   spring.data.neo4j.password=admin
6
7
8   # aws
9   #server.port=5000
10  #spring.data.neo4j.uri=bolt://52.58.238.130:7687
11  #spring.data.neo4j.username=neo4j
12  #spring.data.neo4j.password=redacted
13
14  amazon.region=eu-central-1
15  amazon.accessKey=redacted
16  amazon.secretKey=redacted
17
```

Figure 7 - Configuration properties of application backend

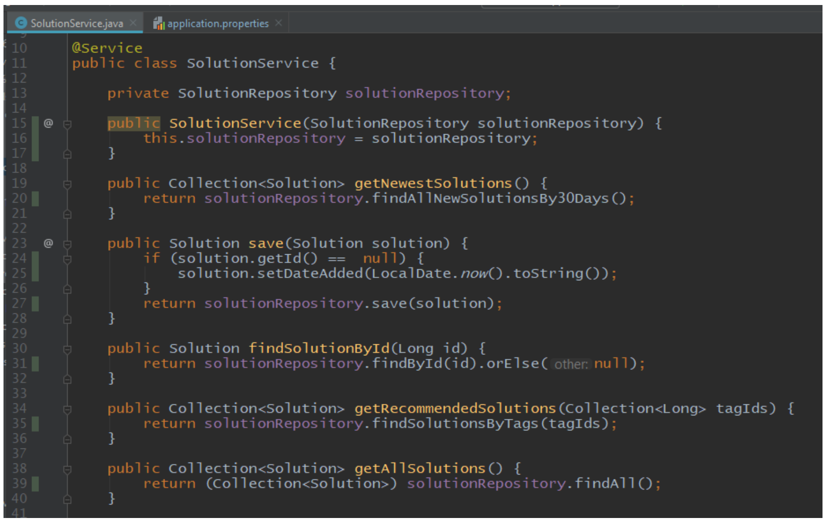
I am connecting to an instance of my database running on a Docker container and connecting to this through Spring. I am running the instance locally and configuring my application to look to the correct port, as shown in Figure 7 on lines 3-5. I have also deployed the database and backend using AWS and have the configuration settings for that commented out on lines 9-12 and 14-15 in Figure 7 (note: lines 12, 15 & 16 credentials are redacted for security reasons).



```
1   package com.cognizant.connect.repository;
2
3   import com.cognizant.connect.domain.Solution;
4   import org.springframework.data.neo4j.annotation.Query;
5   import org.springframework.data.neo4j.repository.Neo4jRepository;
6   import org.springframework.stereotype.Repository;
7
8   import java.util.Collection;
9
10  @Repository
11  public interface SolutionRepository extends Neo4jRepository<Solution, Long> {
12
13      @Query("MATCH (s:Solution) WHERE s.dateAdded >= (toString(date() - duration('P30D'))) AND NOT s.phase = 'Sensed' RETURN
14      Collection<Solution> findAllNewSolutionsBy30Days();
15
16      @Query("WITH {0} as tagIds MATCH (t:Tag)-[:TAGGED]->(s:Solution) WHERE id(t) in tagIds AND NOT s.phase = 'Sensed' RETUR
17      Collection<Solution> findSolutionsByTags(Collection<Long> tagIds);
18
19      @Query("MATCH (s:Solution) WHERE s.name =~ ('(?i)' + {0}) RETURN s")
20      Collection<Solution> searchOnName(String names);
21
22      @Query("MATCH (u:User), (s:Solution) WHERE u.name = {0} AND id(s) = {1} CREATE (u)-[:LAST_MODIFIED_BY]->(s)")
23      void addLastModifiedByUser(String userName, Long solutionId);
24
25      @Query("MATCH (u:User)-[r:LAST_MODIFIED_BY]->(s:Solution) WHERE u.name = {0} AND id(s) = {1} DELETE r")
26      void deleteLastModifiedByUser(String userName, Long solutionId);
27
28      @Query("MATCH (u:User), (s:Solution) WHERE u.name = {0} AND id(s) = {1} CREATE (u)-[:SENSED_BY]->(s)")
29      void addSensedBy(String userName, Long solutionId);
30
31  }
32
```

Figure 8 - Solution Repository

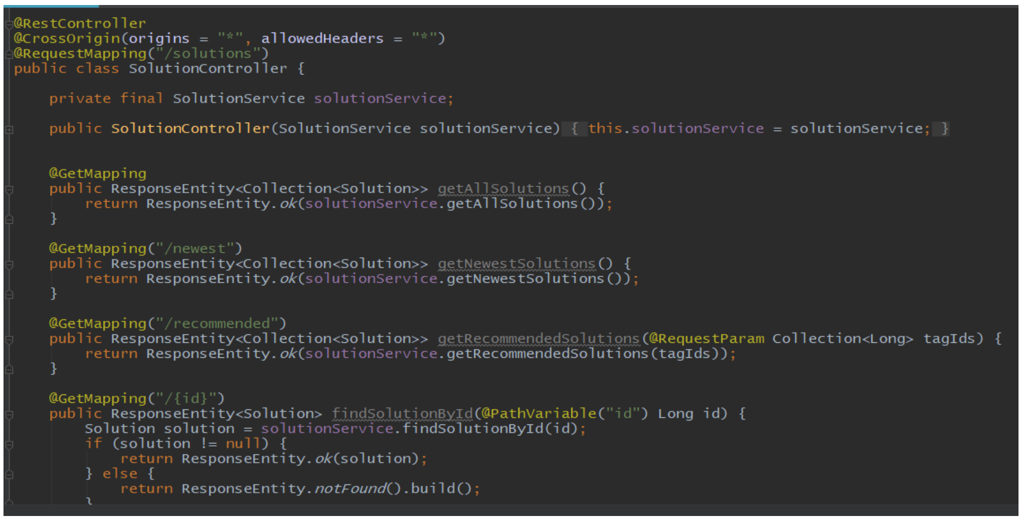
Figure 8 shows the repository layer. I am using an interface which is extending the already existing Neo4j repository. I am using this because I would like to use the in-built methods that the Neo4j repository has, as well as writing my own custom queries. Line 13 in Figure 8 demonstrates an in built query, written in the Cypher, the graph query language.



```
@Service
public class SolutionService {

    private SolutionRepository solutionRepository;

    public SolutionService(SolutionRepository solutionRepository) {
        this.solutionRepository = solutionRepository;
    }

    public Collection<Solution> getNewestSolutions() {
        return solutionRepository.findAllNewSolutionsBy30Days();
    }

    public Solution save(Solution solution) {
        if (solution.getId() ==  null) {
            solution.setDateAdded(LocalDate.now().toString());
        }
        return solutionRepository.save(solution);
    }

    public Solution findSolutionById(Long id) {
        return solutionRepository.findById(id).orElse( other: null);
    }

    public Collection<Solution> getRecommendedSolutions(Collection<Long> tagIds) {
        return solutionRepository.findSolutionsByTags(tagIds);
    }

    public Collection<Solution> getAllSolutions() {
        return (Collection<Solution>) solutionRepository.findAll();
    }

```

Figure 9 - Solution Service

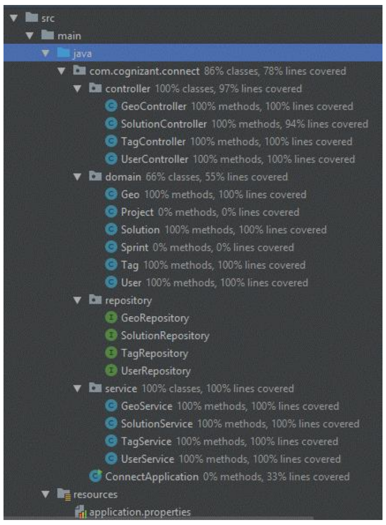
Figure 9 shows part of the service layer class, which connects to the repository layer and performs all the 'logic'. The method names clearly show what each method does, the methods are short with only one or a few lines. This demonstrates I have written clean quality code, which is easy to read and understand, which is important for future developers who work on the code to be able interpret the code easier.



```java
@RestController
@CrossOrigin(origins = "*", allowedHeaders = "*")
@RequestMapping("/solutions")
public class SolutionController {

    private final SolutionService solutionService;

    public SolutionController(SolutionService solutionService) { this.solutionService = solutionService; }


    @GetMapping
    public ResponseEntity<Collection<Solution>> getAllSolutions() {
        return ResponseEntity.ok(solutionService.getAllSolutions());
    }

    @GetMapping("/newest")
    public ResponseEntity<Collection<Solution>> getNewestSolutions() {
        return ResponseEntity.ok(solutionService.getNewestSolutions());
    }

    @GetMapping("/recommended")
    public ResponseEntity<Collection<Solution>> getRecommendedSolutions(@RequestParam Collection<Long> tagIds) {
        return ResponseEntity.ok(solutionService.getRecommendedSolutions(tagIds));
    }

    @GetMapping("/{id}")
    public ResponseEntity<Solution> findSolutionById(@PathVariable("id") Long id) {
        Solution solution = solutionService.findSolutionById(id);
        if (solution != null) {
            return ResponseEntity.ok(solution);
        } else {
            return ResponseEntity.notFound().build();
```

Figure 10 - Solution Controller

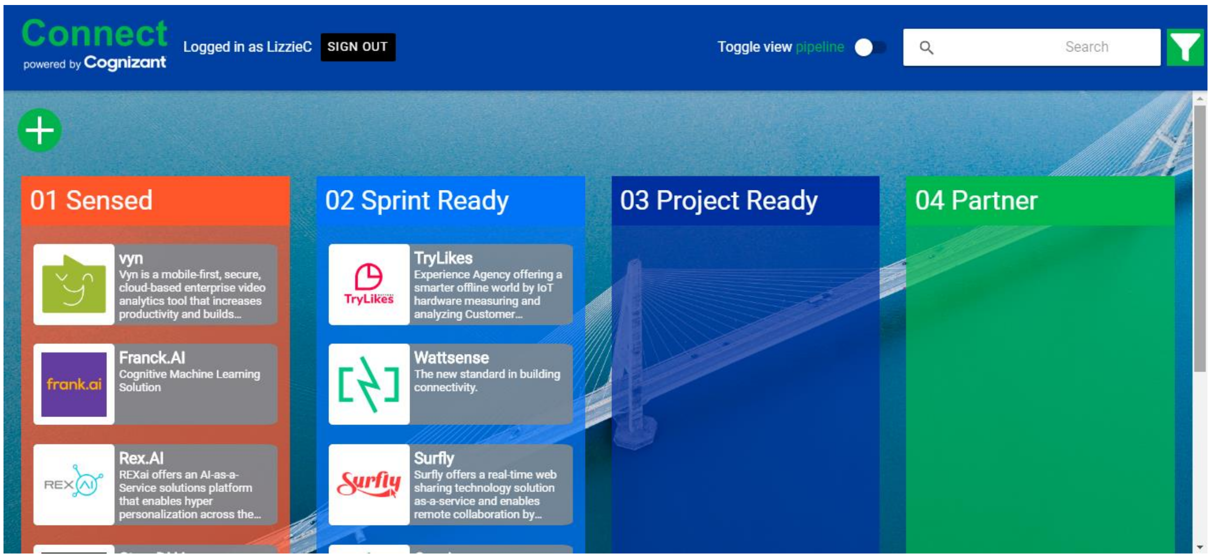
Figure 10 shows the controller layer. This is annotated as a REST controller as we want to perform CRUD operations. All figures 4-7 demonstrate that I have linked the database to my backend code and can execute a RESTful service with CRUD statements and other custom queries. This is also demonstrates object oriented practice, as each layer is a separate class having one responsibility, and that one data type (Solution in this case) is also separated into its own classes and dependency inject layers shown below in Figure 11.



```
▼ src
  ▼ main
    ▼ java
      ▼ com.cognizant.connect 86% classes, 78% lines covered
        ▼ controller 100% classes, 97% lines covered
            GeoController 100% methods, 100% lines covered
            SolutionController 100% methods, 94% lines covered
            TagController 100% methods, 100% lines covered
            UserController 100% methods, 100% lines covered
        ▼ domain 66% classes, 55% lines covered
            Geo 100% methods, 100% lines covered
            Project 0% methods, 0% lines covered
            Solution 100% methods, 100% lines covered
            Sprint 0% methods, 0% lines covered
            Tag 100% methods, 100% lines covered
            User 100% methods, 100% lines covered
        ▼ repository
            GeoRepository
            SolutionRepository
            TagRepository
            UserRepository
        ▼ service 100% classes, 100% lines covered
            GeoService 100% methods, 100% lines covered
            SolutionService 100% methods, 100% lines covered
            TagService 100% methods, 100% lines covered
            UserService 100% methods, 100% lines covered
          ConnectApplication 0% methods, 33% lines covered
    ▼ resources
        application.properties
```

Figure 11 - Backend code folder structure

Frontend



vyn
Vyn is a mobile-first, secure, cloud-based enterprise video analytics tool that increases productivity and builds...

Franck.AI
Cognitive Machine Learning Solution

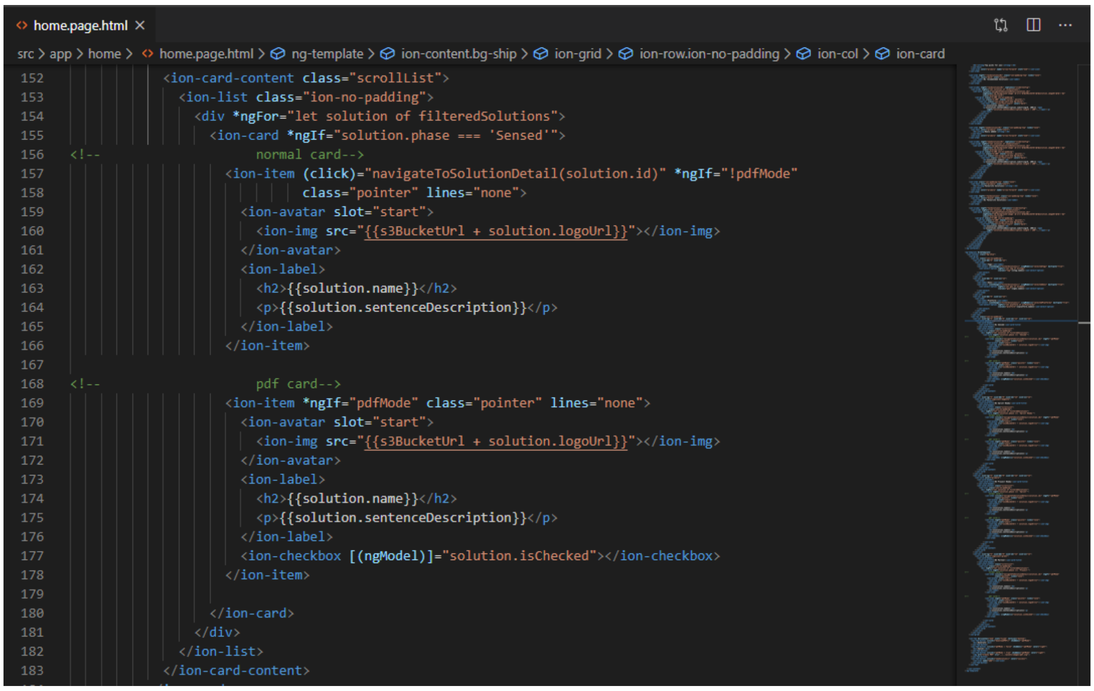
Rex.AI
REXai offers an AI-as-a-Service solutions platform that enables hyper personalization across the...

TryLikes
Experience Agency offering a smarter offline world by IoT hardware measuring and analyzing Customer...

Wattsense
The new standard in building connectivity.

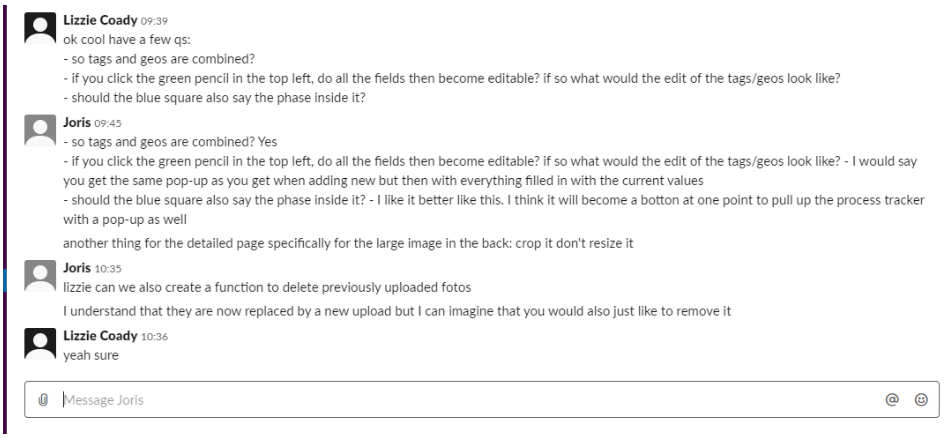
Surfly
Surfly offers a real-time web sharing technology solution as-a-service and enables remote collaboration by...

Figure 12 - App home page UI


home.page.html
src > app > home > home.page.html > ng-template > ion-content.bg-ship > ion-grid > ion-row.ion-no-padding > ion-col > ion-card

```
            <ion-card-content class="scrollList">
              <ion-list class="ion-no-padding">
                <div *ngFor="let solution of filteredSolutions">
                  <ion-card *ngIf="solution.phase === 'Sensed'">
  <!--                   normal card-->
                    <ion-item (click)="navigateToSolutionDetail(solution.id)" *ngIf="!pdfMode"
                              class="pointer" lines="none">
                      <ion-avatar slot="start">
                        <ion-img src="{{s3BucketUrl + solution.logoUrl}}"></ion-img>
                      </ion-avatar>
                      <ion-label>
                        <h2>{{solution.name}}</h2>
                        <p>{{solution.sentenceDescription}}</p>
                      </ion-label>
                    </ion-item>

  <!--                   pdf card-->
                    <ion-item *ngIf="pdfMode" class="pointer" lines="none">
                      <ion-avatar slot="start">
                        <ion-img src="{{s3BucketUrl + solution.logoUrl}}"></ion-img>
                      </ion-avatar>
                      <ion-label>
                        <h2>{{solution.name}}</h2>
                        <p>{{solution.sentenceDescription}}</p>
                      </ion-label>
                      <ion-checkbox [(ngModel)]="solution.isChecked"></ion-checkbox>
                    </ion-item>

                  </ion-card>
                </div>
              </ion-list>
            </ion-card-content>
```

Figure 13 - Home Page HTML portion of code


Lizzie Coady 09:39
ok cool have a few qs:
- so tags and geos are combined?
- if you click the green pencil in the top left, do all the fields then become editable? if so what would the edit of the tags/geos look like?
- should the blue square also say the phase inside it?

Joris 09:45
- so tags and geos are combined? Yes
- if you click the green pencil in the top left, do all the fields then become editable? if so what would the edit of the tags/geos look like? - I would say you get the same pop-up as you get when adding new but then with everything filled in with the current values
- should the blue square also say the phase inside it? - I like it better like this. I think it will become a botton at one point to pull up the process tracker with a pop-up as well

another thing for the detailed page specifically for the large image in the back: crop it don't resize it

Joris 10:35
lizzie can we also create a function to delete previously uploaded fotos

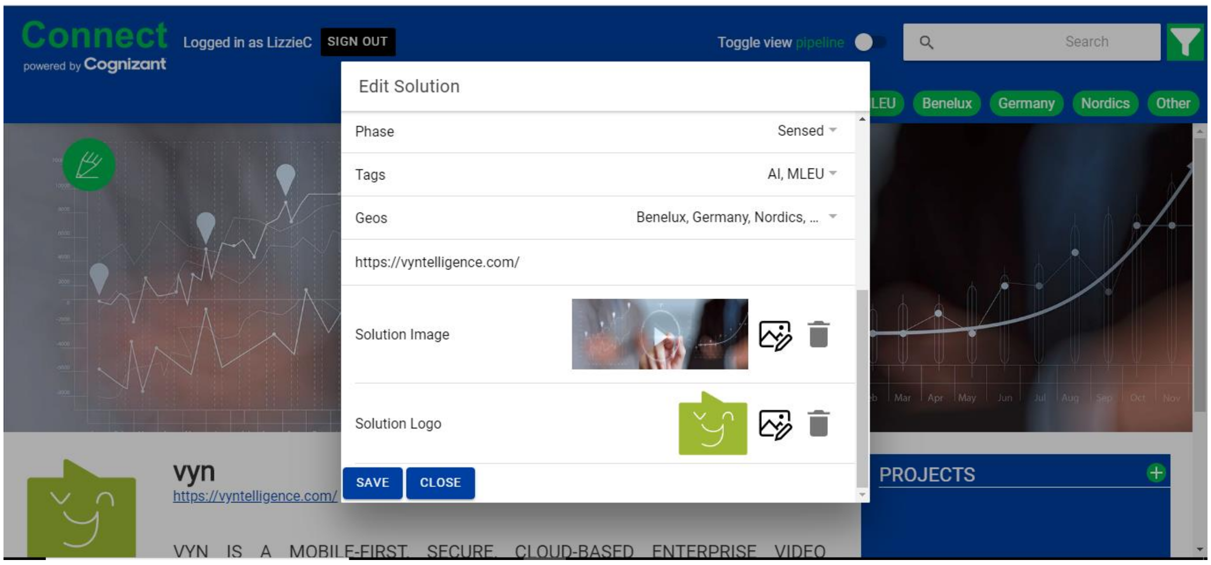
I understand that they are now replaced by a new upload but I can imagine that you would also just like to remove it

Lizzie Coady 10:36
yeah sure

Message Joris

Figure 14 - Slack messages between me and Joris, the product owner and manager

Figure 12 shows the homepage of the app and Figure 13 shows the respective portion of the HTML code for the page. I wrote the whole of the code in Figure 13 and had a contribution along with two of my colleagues in the design of it show in Figure 12. The section of code shown in Figure 13 creates the orange column shown in Figure 12. Line 154 starts the for loop, looping through a list of Solution type objects, which are the further filtered on a conditional statement in line 155, creating each of the 3 grey blocks with the rest of the lines. The homepage has been made like this as the users wanted to be able to see the solutions they have entered, with what stage they are at (e.g. sensed, sprint ready) being the first thing they see, which was agreed from testing that this was the most important. I have daily correspondence with Joris who I work on the platform with where we discuss any queries, further improvements and generally to align as were work remotely. Figure 14 demonstrates an example of the usual conversations, this one requesting a simple feature that we had overlooked to implement. Working like this means we stay aligned and continue working in an agile way. We have daily stand ups with the whole team where any queries from me and the others can also get raised. I have demonstrated this by answering the feature requesting of deleting an uploaded image show in Figure 15, where the grey bin icon can be clicked to remove the uploaded image from the page.



Edit Solution

Phase: Sensed
Tags: AI, MLEU
Geos: Benelux, Germany, Nordics, ...
https://vyntelligence.com/
Solution Image
Solution Logo

Figure 15 - 'Edit solution' app page

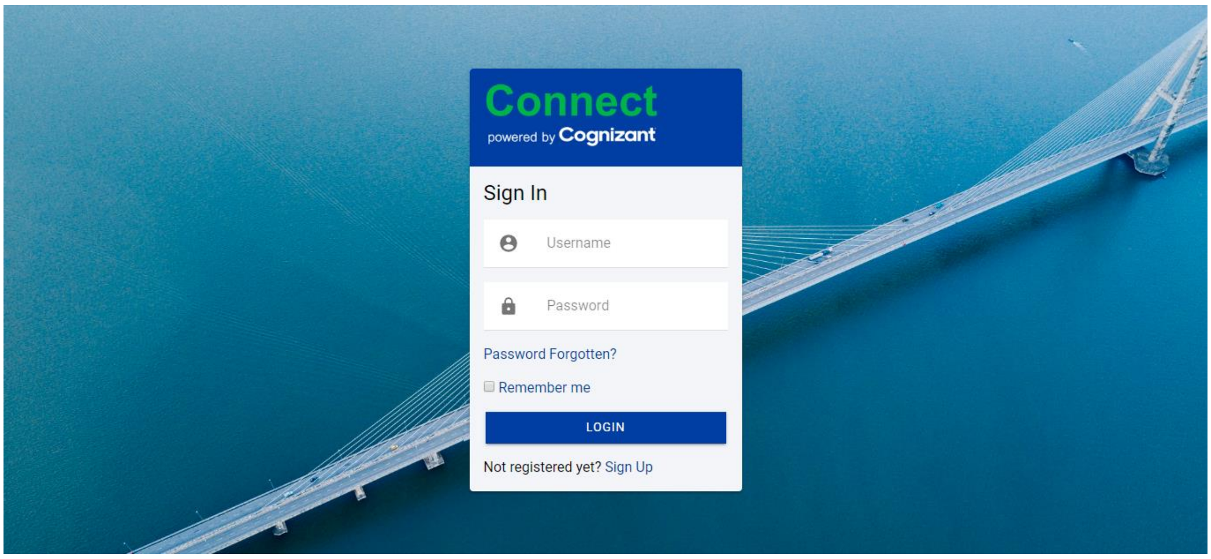
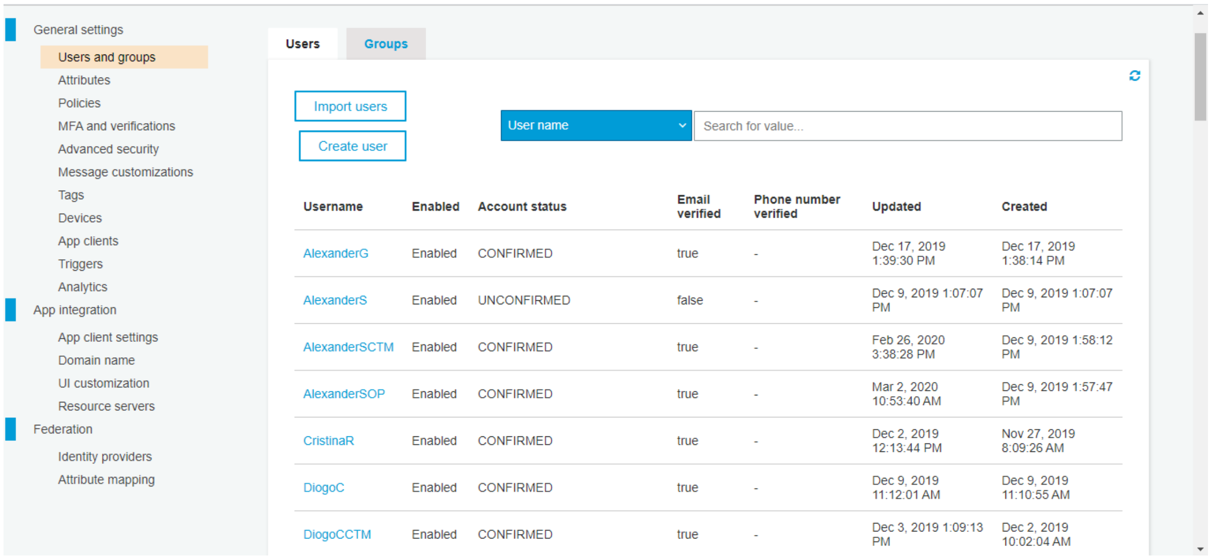
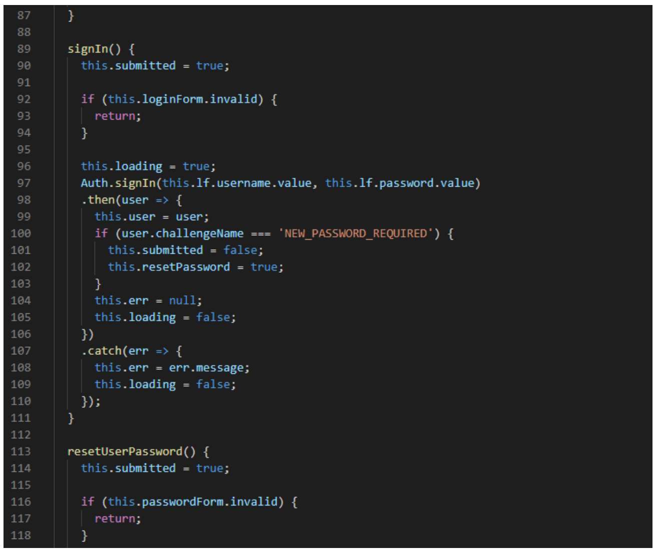


Figure 16 – App login screen



Username	Enabled	Account status	Email verified	Phone number verified	Updated	Created
AlexanderG	Enabled	CONFIRMED	true	-	Dec 17, 2019 1:39:30 PM	Dec 17, 2019 1:38:14 PM
AlexanderS	Enabled	UNCONFIRMED	false	-	Dec 9, 2019 1:07:07 PM	Dec 9, 2019 1:07:07 PM
AlexanderSCTM	Enabled	CONFIRMED	true	-	Feb 26, 2020 3:38:28 PM	Dec 9, 2019 1:58:12 PM
AlexanderSOP	Enabled	CONFIRMED	true	-	Mar 2, 2020 10:53:40 AM	Dec 9, 2019 1:57:47 PM
CristinaR	Enabled	CONFIRMED	true	-	Dec 2, 2019 12:13:44 PM	Nov 27, 2019 8:09:26 AM
DiogoC	Enabled	CONFIRMED	true	-	Dec 9, 2019 11:12:01 AM	Dec 9, 2019 11:10:55 AM
DiogoCCTM	Enabled	CONFIRMED	true	-	Dec 3, 2019 1:09:13 PM	Dec 2, 2019 10:02:04 AM

Figure 17 - Cognito user management screen



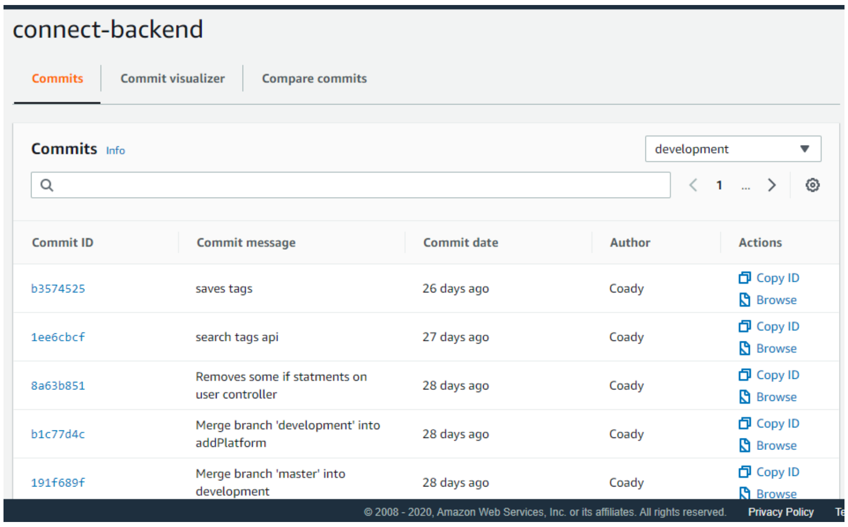
```
 87    }
 88
 89    signIn() {
 90      this.submitted = true;
 91
 92      if (this.loginForm.invalid) {
 93        return;
 94      }
 95
 96      this.loading = true;
 97      Auth.signIn(this.lf.username.value, this.lf.password.value)
 98      .then(user => {
 99        this.user = user;
100        if (user.challengeName === 'NEW_PASSWORD_REQUIRED') {
101          this.submitted = false;
102          this.resetPassword = true;
103        }
104        this.err = null;
105        this.loading = false;
106      })
107      .catch(err => {
108        this.err = err.message;
109        this.loading = false;
110      });
111    }
112
113    resetUserPassword() {
114      this.submitted = true;
115
116      if (this.passwordForm.invalid) {
117        return;
118      }
```

Figure 18 - Login page component code

Another component I built to be used in the app was the sign up and login, the UI for this shown in Figure 16. The screen shows a standard login screen with username and password fields with a button to have the option to sign up as a user. This would enable us to use secure login and sign up for new users and then be able to differentiate between the two user groups. Looking back in Figure 2, we were inclined to go more towards a single sign on approach, but after having a few meetings with the Cognizant team who manages our internal systems, we realised that this would be a lengthy and complicated process to implement SSO, so we changed our user story to have users be able to register, having no link to their Cognizant account. We decided to have this feature for after the MVP. I created a basic activity diagram (Figure 3) of the sign up process we would be using, showing that we would be using a username and email address parameter so we can verify the user. This would be useful as well as we had two user groups which would need to be differentiated upon login to provide different UI screens, the diagram showing that it checks whether some is an internal Connect Team Member (CTM) or an external Orchestration Partner (OP).

As we were using AWS services, I had had brief experience in using the user management service Cognito and thought we could use it in the application. I planned that we would have registered users on Cognito show in Figure 17, as it makes more sense to use a reliable and secure user management system rather than creating one myself, and then store the corresponding users in our graph database, with only the username to match the two corresponding users, so we can have user objects to create relationships with our other data types such as solutions, tags, geos etc. This ensures that we have a secure security system to protect all Cognizant employees and external clients who use the app, showing that I have thought about the company's practices and security. I used the AWS frontend library Amplify for the Cognito connection as show in Figure 18. Line 89 in the code shows the start of the sign in function, with line 97 incorporating the Amplify library's sign in API to handle the process. This whole process also ensured that no unverified user could access the app either, as we also had an auth guard service that would redirect anyone who isn't signed in back to the login screen.

Version Control



connect-backend

Commits | Commit visualizer | Compare commits

Commits Info — development

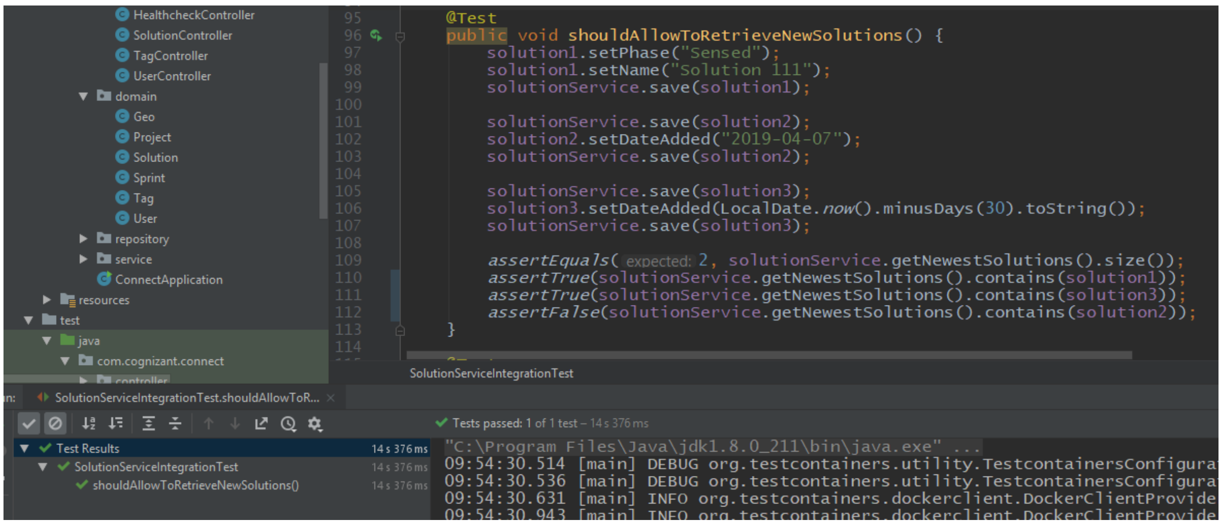
Commit ID	Commit message	Commit date	Author	Actions
b3574525	saves tags	26 days ago	Coady	Copy ID Browse
1ee6cbcf	search tags api	27 days ago	Coady	Copy ID Browse
8a63b851	Removes some if statments on user controller	28 days ago	Coady	Copy ID Browse
b1c77d4c	Merge branch 'development' into addPlatform	28 days ago	Coady	Copy ID Browse
191f689f	Merge branch 'master' into development	28 days ago	Coady	Copy ID Browse

© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy

Figure 19 - The AWS service CodeCommit used for the code repositories showing connect-backend

All codebases I have made are stored on the AWS service CodeCommit. Figure 19 shows a portion of the most recent commits by me from the development branch of the backend code. Like common developer practice, I store the backend and frontend code in online repositories to manage version control and for new developers to be able to pull and develop the code.

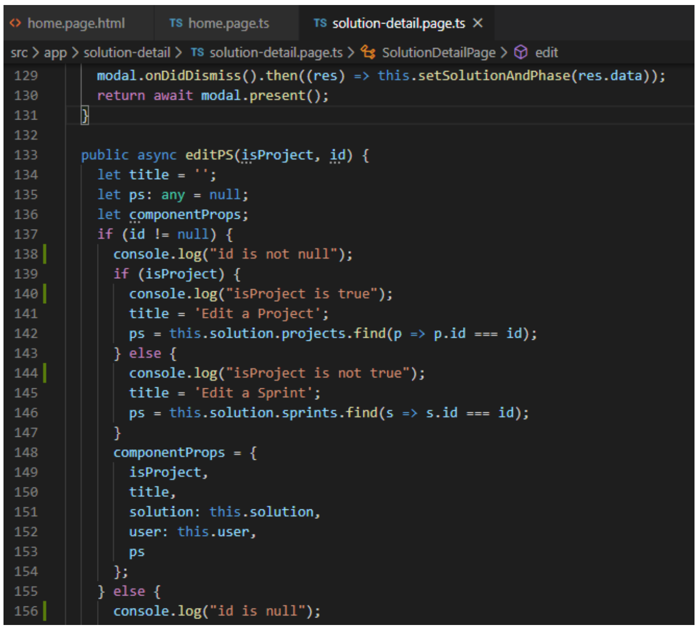
Testing & Debugging



```java
    @Test
    public void shouldAllowToRetrieveNewSolutions() {
        solution1.setPhase("Sensed");
        solution1.setName("Solution 111");
        solutionService.save(solution1);

        solutionService.save(solution2);
        solution2.setDateAdded("2019-04-07");
        solutionService.save(solution2);

        solutionService.save(solution3);
        solution3.setDateAdded(LocalDate.now().minusDays(30).toString());
        solutionService.save(solution3);

        assertEquals( expected: 2, solutionService.getNewestSolutions().size());
        assertTrue(solutionService.getNewestSolutions().contains(solution1));
        assertTrue(solutionService.getNewestSolutions().contains(solution3));
        assertFalse(solutionService.getNewestSolutions().contains(solution2));
    }
```

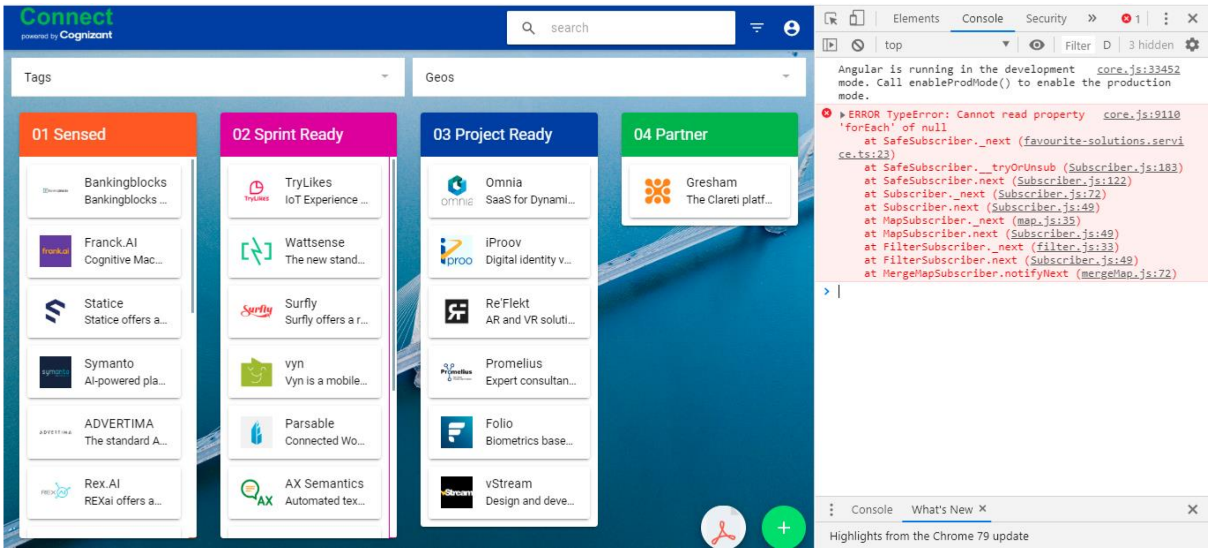
```
✓ Tests passed: 1 of 1 test – 14 s 376 ms

▼ ✓ Test Results                                   14 s 376 ms
  ▼ ✓ SolutionServiceIntegrationTest                14 s 376 ms
      ✓ shouldAllowToRetrieveNewSolutions()         14 s 376 ms

"C:\Program Files\Java\jdk1.8.0_211\bin\java.exe" ...
09:54:30.514 [main] DEBUG org.testcontainers.utility.TestcontainersConfigura
09:54:30.536 [main] DEBUG org.testcontainers.utility.TestcontainersConfigura
09:54:30.631 [main] INFO org.testcontainers.dockerclient.DockerClientProvide
09:54:30.943 [main] INFO org.testcontainers.dockerclient.DockerClientProvide
```

Figure 20 - A unit test testing to retrieve all newest solutions (solutions that have been added 30 days or less ago)


home.page.html
home.page.ts
solution-detail.page.ts
src > app > solution-detail > solution-detail.page.ts > SolutionDetailPage > edit

```
    modal.onDidDismiss().then((res) => this.setSolutionAndPhase(res.data));
    return await modal.present();
  }

  public async editPS(isProject, id) {
    let title = '';
    let ps: any = null;
    let componentProps;
    if (id != null) {
      console.log("id is not null");
      if (isProject) {
        console.log("isProject is true");
        title = 'Edit a Project';
        ps = this.solution.projects.find(p => p.id === id);
      } else {
        console.log("isProject is not true");
        title = 'Edit a Sprint';
        ps = this.solution.sprints.find(s => s.id === id);
      }
      componentProps = {
        isProject,
        title,
        solution: this.solution,
        user: this.user,
        ps
      };
    } else {
      console.log("id is null");
```

Figure 21 - Front end code section of solution detail page showing a function to edit projects and sprints

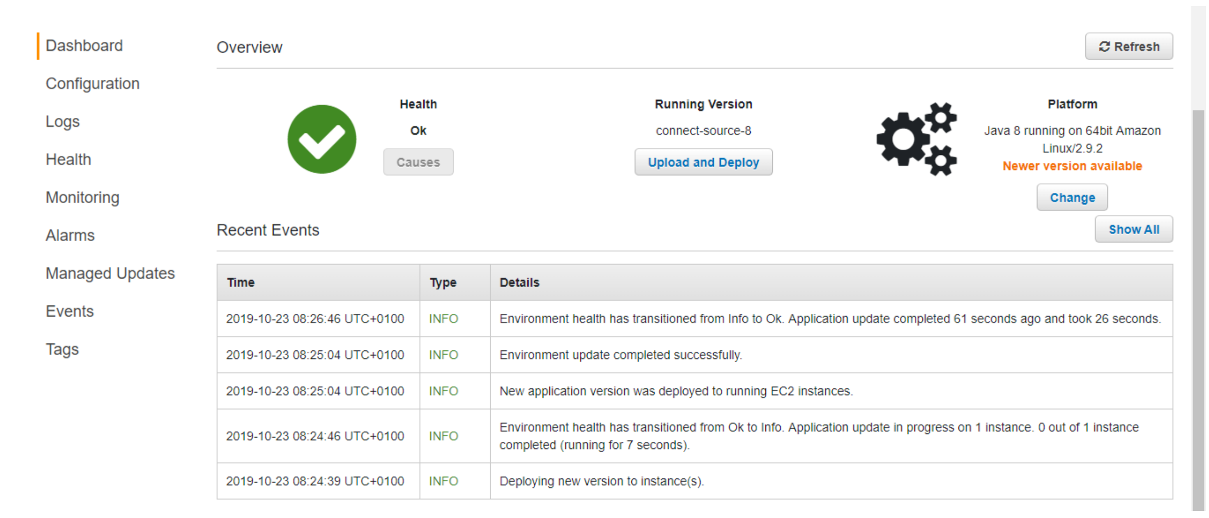


```
Angular is running in the development   core.js:33452
mode. Call enableProdMode() to enable the production
mode.
▸ERROR TypeError: Cannot read property   core.js:9110
'forEach' of null
    at SafeSubscriber._next (favourite-solutions.servi
ce.ts:23)
    at SafeSubscriber.__tryOrUnsub (Subscriber.js:183)
    at SafeSubscriber.next (Subscriber.js:122)
    at Subscriber._next (Subscriber.js:72)
    at Subscriber.next (Subscriber.js:49)
    at MapSubscriber._next (map.js:35)
    at MapSubscriber.next (Subscriber.js:49)
    at FilterSubscriber._next (filter.js:33)
    at FilterSubscriber.next (Subscriber.js:49)
    at MergeMapSubscriber.notifyNext (mergeMap.js:72)
> |
```

Figure 22 - Developer console open on home page of app displaying an error message

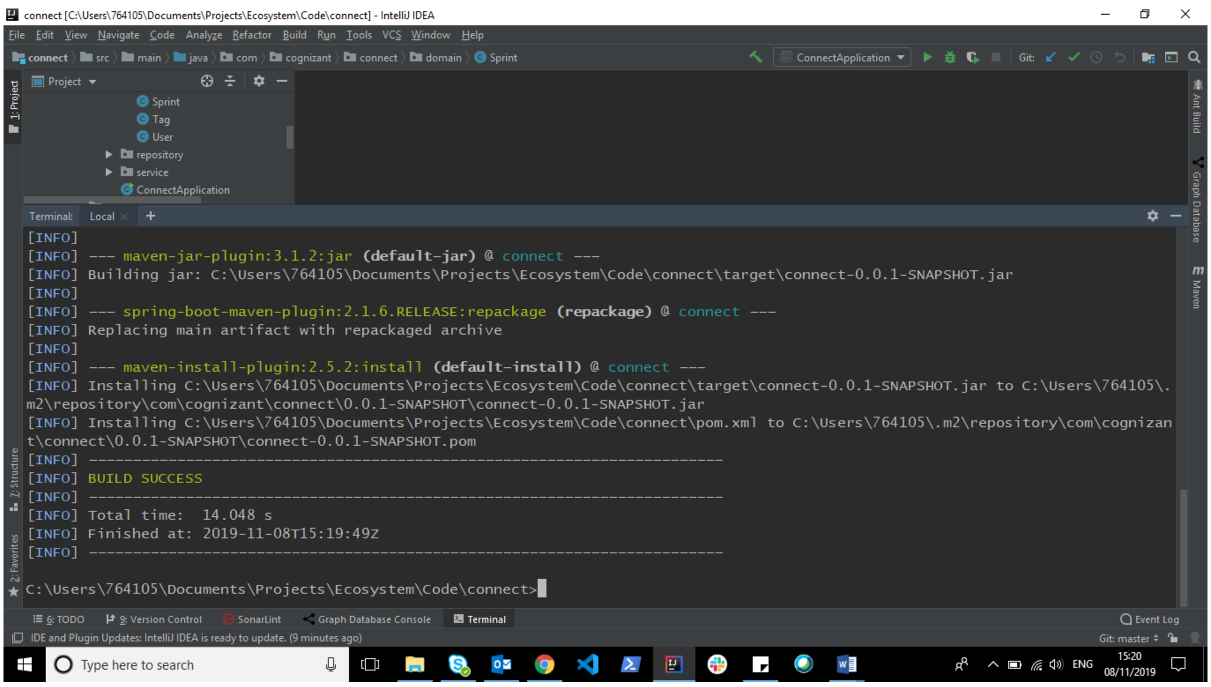
The coding process I used also included using unit tests to test my code, Figure 20 demonstrating this. I am testing the service for getting all newest solutions, which are solutions that have been added 30 days or less ago. I want to test where the code will meet this criteria, so I have created it so I have 2 solution examples (solution1 and solution3 in Figure 20) where I have set the 'dateAdded' field to dates that I know fit the criteria (solution1 having today's date solution3 having today's date minus 30 days), and 1 solution example which is set to more than 30 days old. I expect therefore the method to return 2 of the 3 solutions (line 109), also further tested the method is returning the exact solutions I expect (lines 110-112). Another process used, particularly in front end coding, was looking to the developer console on the chrome browser. If the code wasn't doing what I expected, I would put comments in each section of the code to see where it would stop, adjust the code then repeat until it worked as expected. This is shown in lines 138, 140, 144 and 156 of Figure 21. I would also look at errors that the framework or the libraries I was using printed to the console to diagnose the issue, then looking in the library documentation and/or googling the error to find a solution. This is displayed in Figure 22 where I can see that a forEach function is trying to be performed on a null object and at what line and file of the code, meaning I can find where the issue is straight away.

Deployment



Time	Type	Details
2019-10-23 08:26:46 UTC+0100	INFO	Environment health has transitioned from Info to Ok. Application update completed 61 seconds ago and took 26 seconds.
2019-10-23 08:25:04 UTC+0100	INFO	Environment update completed successfully.
2019-10-23 08:25:04 UTC+0100	INFO	New application version was deployed to running EC2 instances.
2019-10-23 08:24:46 UTC+0100	INFO	Environment health has transitioned from Ok to Info. Application update in progress on 1 instance. 0 out of 1 instance completed (running for 7 seconds).
2019-10-23 08:24:39 UTC+0100	INFO	Deploying new version to instance(s).

Figure 23 - Elastic beanstalk dashboard of backend deployment


connect [C:\Users\764105\Documents\Projects\Ecosystem\Code\connect] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
connect src main java com cognizant connect domain Sprint
ConnectApplication
Project
Sprint
Tag
User
repository
service
ConnectApplication
Terminal: Local
TODO
Version Control
SonarLint
Graph Database Console
Terminal
Event Log
IDE and Plugin Updates: IntelliJ IDEA is ready to update. (9 minutes ago)
Git: master
Type here to search
ENG
15:20
08/11/2019

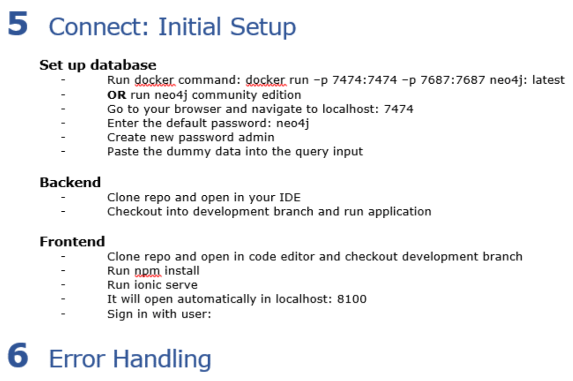
```
[INFO]
[INFO] --- maven-jar-plugin:3.1.2:jar (default-jar) @ connect ---
[INFO] Building jar: C:\Users\764105\Documents\Projects\Ecosystem\Code\connect\target\connect-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.1.6.RELEASE:repackage (repackage) @ connect ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ connect ---
[INFO] Installing C:\Users\764105\Documents\Projects\Ecosystem\Code\connect\target\connect-0.0.1-SNAPSHOT.jar to C:\Users\764105\.
m2\repository\com\cognizant\connect\0.0.1-SNAPSHOT\connect-0.0.1-SNAPSHOT.jar
[INFO] Installing C:\Users\764105\Documents\Projects\Ecosystem\Code\connect\pom.xml to C:\Users\764105\.m2\repository\com\cognizan
t\connect\0.0.1-SNAPSHOT\connect-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  14.048 s
[INFO] Finished at: 2019-11-08T15:19:49Z
[INFO] ------------------------------------------------------------------------

C:\Users\764105\Documents\Projects\Ecosystem\Code\connect>
```

Figure 24 - mvn install command to build jar file

For first time deployment, I used Elastic Beanstalk from AWS. Elastic Beanstalk is a service that makes it very simple to deploy an application. Firstly, I created an environment and followed all the default steps (Figure 23 shows built environment's dashboard). You now need to deploy the code to the environment. This environment will need to connect to our production database and not our local one, so some configurations need to be changed before deployment. To create the jar file to use, I first switched the configuration properties from local to the deployed database properties (Figure 4, uncommenting lines 9-12 and commenting lines 2-5), then ran 'mvn install' (Figure 24) to build the jar. This demonstrates I am able to deploy code to production environments by setting up correct configuration properties to be able make the deployment process work correctly and be used by my team members.

While working to deploy the application, we considered the small budget of the project and that we would want to keep the costs as minimal as possible. As we knew a very small number of people would be using the app initially (up to 10), we chose one of the smallest instance size of a t2.micro. We would have then up scaled this when we released the application to Cognizant employees and clients.

Review



5 Connect: Initial Setup

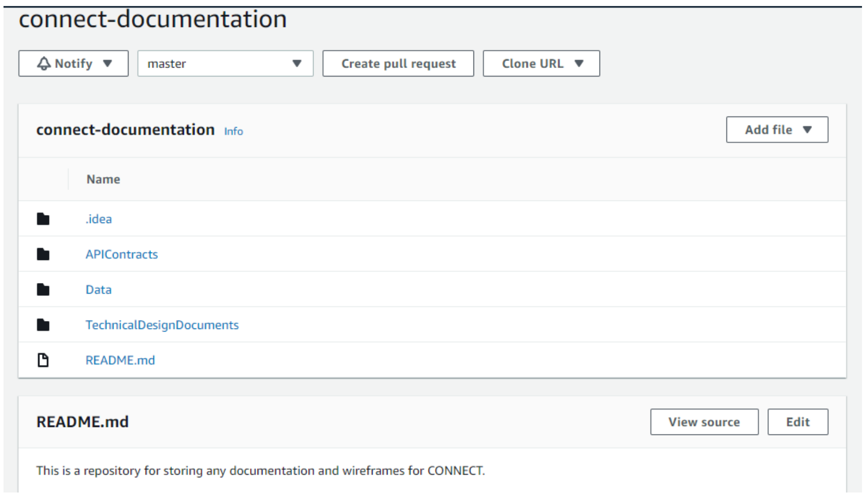
Set up database
- Run docker command: docker run –p 7474:7474 –p 7687:7687 neo4j: latest
- **OR** run neo4j community edition
- Go to your browser and navigate to localhost: 7474
- Enter the default password: neo4j
- Create new password admin
- Paste the dummy data into the query input

Backend
- Clone repo and open in your IDE
- Checkout into development branch and run application

Frontend
- Clone repo and open in code editor and checkout development branch
- Run npm install
- Run ionic serve
- It will open automatically in localhost: 8100
- Sign in with user:

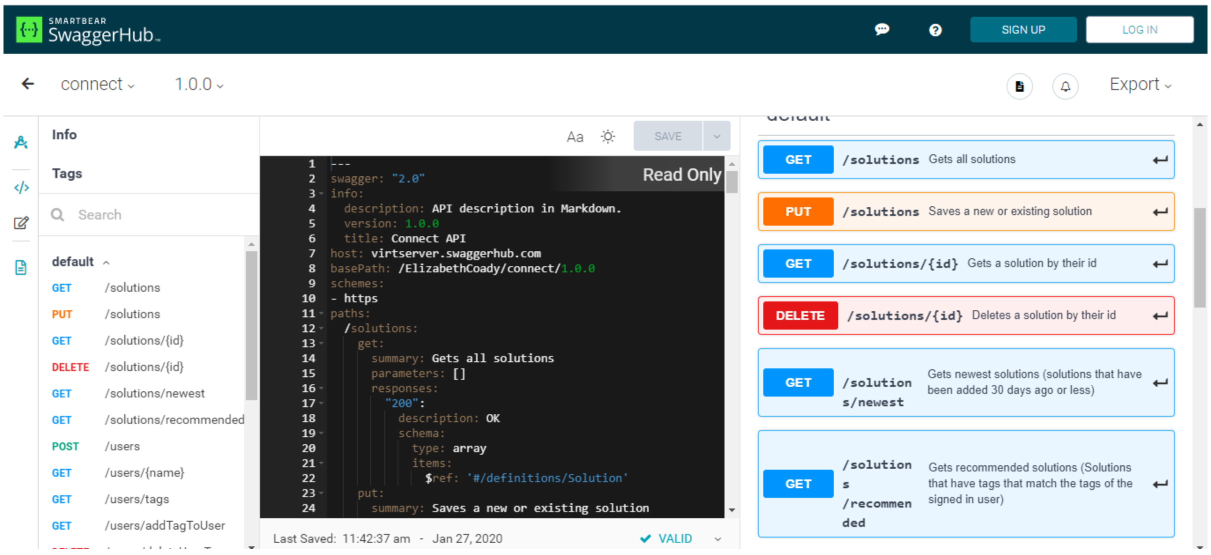
6 Error Handling

Figure 25 - Screenshot of user documentation provided for first time set up



This is a repository for storing any documentation and wireframes for CONNECT.

Figure 26 - Documentation repository on AWS CodeCommit



default

GET	/solutions
PUT	/solutions
GET	/solutions/{id}
DELETE	/solutions/{id}
GET	/solutions/newest
GET	/solutions/recommended
POST	/users
GET	/users/{name}
GET	/users/tags
GET	/users/addTagToUser

```yaml
---
swagger: "2.0"
info:
  description: API description in Markdown.
  version: 1.0.0
  title: Connect API
host: virtserver.swaggerhub.com
basePath: /ElizabethCoady/connect/1.0.0
schemes:
- https
paths:
  /solutions:
    get:
      summary: Gets all solutions
      parameters: []
      responses:
        "200":
          description: OK
          schema:
            type: array
            items:
              $ref: '#/definitions/Solution'
    put:
      summary: Saves a new or existing solution
```

Last Saved: 11:42:37 am - Jan 27, 2020 ✓ VALID

GET	/solutions	Gets all solutions
PUT	/solutions	Saves a new or existing solution
GET	/solutions/{id}	Gets a solution by their id
DELETE	/solutions/{id}	Deletes a solution by their id
GET	/solutions/newest	Gets newest solutions (solutions that have been added 30 days ago or less)
GET	/solutions/recommended	Gets recommended solutions (Solutions that have tags that match the tags of the signed in user)

Figure 27 - Screenshot of Swagger showing part of the API contract

In conclusion, this has been a challenging but extremely rewarding project. I was very grateful to have been given the opportunity to develop an entire project from its beginning essentially. I have worked from planning with Alex, Diogo and Joris about the vision and feasibility of the project, to deploying to production for the use of my team and external clients. As I am still very inexperienced, I have learned how to code dryer through new iterations and through looking at the most recent tutorials and guides to keep up with modern standards of working. This came very important as in the last month we on boarded two software developer apprentices (Alex and Diogo) to help develop the project, where I had to make sure the code was able to be followed by new developers and easy to be picked up and worked on. I needed therefore to develop some user documentation on the setup seen on Figure 25, the API contract for the application shown in Figure 27 with the repository for this on CodeCommit shown in Figure 26. This presented a challenge as I had to also essentially be a technical lead and steer the project, being the only other technical person. It was definitely something out of my comfort zone and a bit anxiety inducing at first, but it turned out really well as we all wanted to perform to our best and make the platform be really amazing and something to be proud of. I have a brief video of me going over the project structure found in the attached video 'ConnectPresentation.mp4'. Overall, this has been an amazing experience being able to delve into all aspects of a development of a project and to work in a really positive and hardworking team, where we are all learning from each other and making ourselves better in our software developing careers.