Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for datateknologi, elektroteknologi og realfag

Visualizing Smart Charging of Electric Vehicles for Support Personnel

System documentation

Version 1.6

REVISION HISTORY

Date	Version	Description	Author
06/MAY/22	1.0	Import system documentation from external sources.	Roger Karlsen Mads Henrik Sørbø
10/MAY/22	1.1	Import project structure from external source.	Roger Karlsen
16/MAY/22	1.2	Import source code documentation.	Roger Karlsen
19/MAY/22	1.3	Various minor fixes.	Roger Karlsen Mads Henrik Sørbø
20/MAY/22	1.4	Obfuscate service names. Update platform and component diagrams.	Roger Karlsen Mads Henrik Sørbø
21/MAY/22	1.5	Write about security.	Kristin Standal
22/MAY/22	1.6	Update architecture text and project structure.	Mads Henrik Sørbø

TABLE OF CONTENTS

1 INTRODUCTION

The purpose of this document is to give a simplified overview of how the smart charging view is built. The smart charging view is a single view in the Varys application. The names of services this view is dependent on, their composition, and endpoints have been obfuscated to protect Tibber’s security. The document contains various models and source code snippets explaining how the various components are structured and how they interact with each other.

2 ARCHITECTURE

The Tibber platform is built as a collection of microservices hosted on Amazon Web Services, using technologies such as lambda functions, events, and queues for interservice communication. Varys is dependent on various other microservices to function. In the case for this project, the most central services are the Customer and Device Orchestrator services (names are obscured for security reasons).

The Device Orchestrator service has information about every device registered on a customer and/or their homes. We call on this endpoint to fetch the devices that are related to the smart charging view. The relevant devices in this case are registered cars, load balancers, and chargers.

The Customer service has information about the customer and their home(s).

Figure 2.1 below shows a simplified overview of the communication between Varys, its backend, and other services. SNS/SQS is a publish-subscribe pattern that uses queues and messages for intercommunication between services.

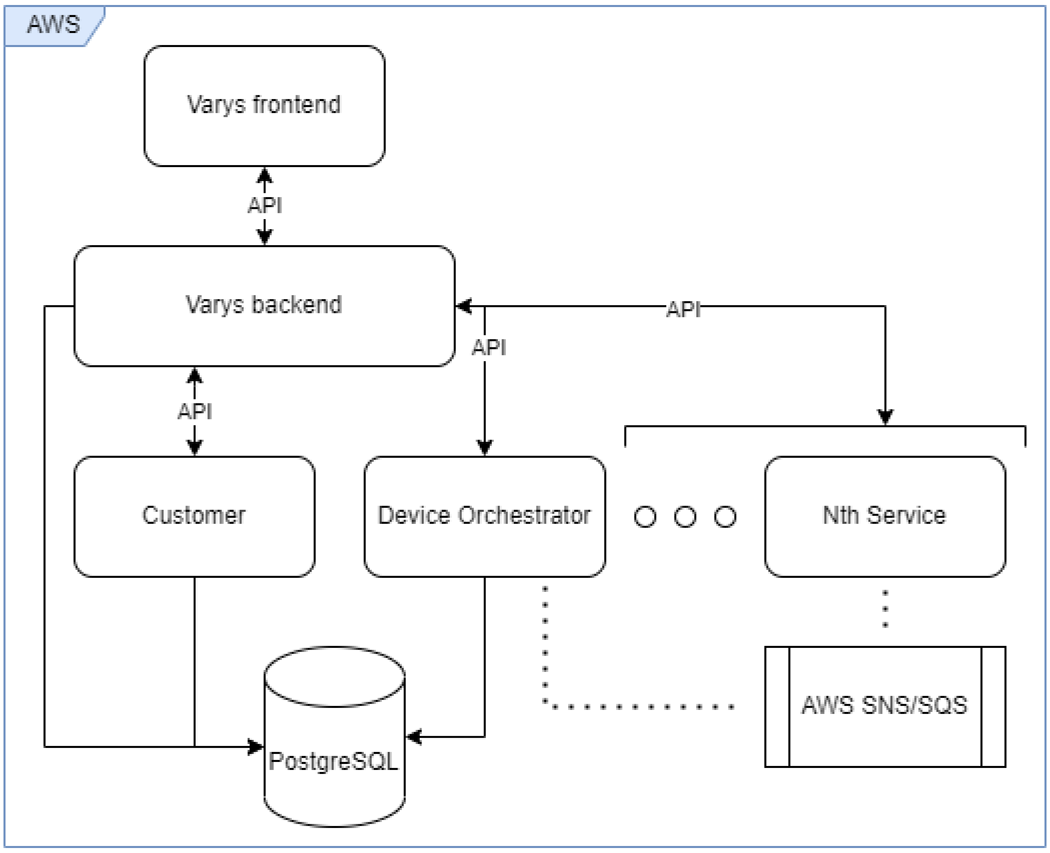


Figure 2.1 Simplified overview of the communication between Varys, its backend, and other services.

As a result of employing a microservices architecture, each service is self-contained and may utilize several technologies that best suit its requirements, such as NetCore, Node JS, PostgreSQL, and Python. In figure 2.2, a data flow generated by NewRelic shows the dependencies between Varys and other microservices in the Tibber-sphere.

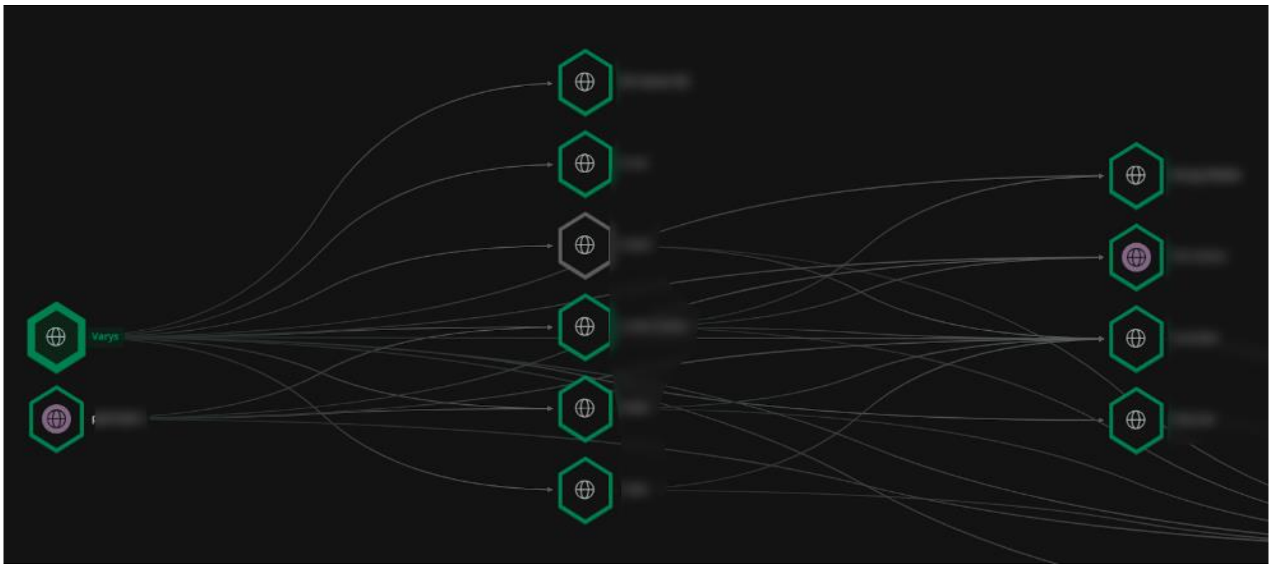


Figure 2.2 Data flow that shows dependencies between Varys and other microservices.

3 PROJECT STRUCTURE

Varys has a client and server component where the server part of the application relies on other microservices. The project structure below shows where the Smart Charging view is located in the Varys application. Almost all of the code written for this project is contained within the “Smart charging”-folder under “Views”. The “smart-charging.js”-file under the “server/routers”-folder is responsible for calling the endpoints and fetching the data needed, so that they can be called in the various components for the Smart Charging view.

```
Varys/
└── src/
    ├── client/
    │   ├── components
    │   ├── filters
    │   ├── mixins
    │   ├── router
    │   ├── styles
    │   ├── utils
    │   ├── views/
    │   │   ├── Other views
    │   │   ├── Smart Charging/
    │   │   │   ├── components/
    │   │   │   │   ├── modals/
    │   │   │   │   ├── AlertItem.vue
    │   │   │   │   ├── AlertItems.vue
    │   │   │   │   ├── BatteryBar.vue
    │   │   │   │   ├── CarsOfflineTable.vue
    │   │   │   │   ├── CarsTable
    │   │   │   │   ├── ChargersTable.vue
    │   │   │   │   ├── ChargingCalculator.vue
    │   │   │   │   ├── HighlightItems.vue
    │   │   │   │   └── LoadBalancingTable.vue
    │   │   │   ├── alert.js
    │   │   │   └── index.vue
    │   │   └── Other views
    │   └── App.vue
    ├── server/
    │   └── routers/
    │       └── smart-charging.js
    └── test/
        ├── client/
        └── server/
```

4 COMPONENT DIAGRAM

Figure 4.1 shows the structure of the most central Vue-components and their inherited properties.

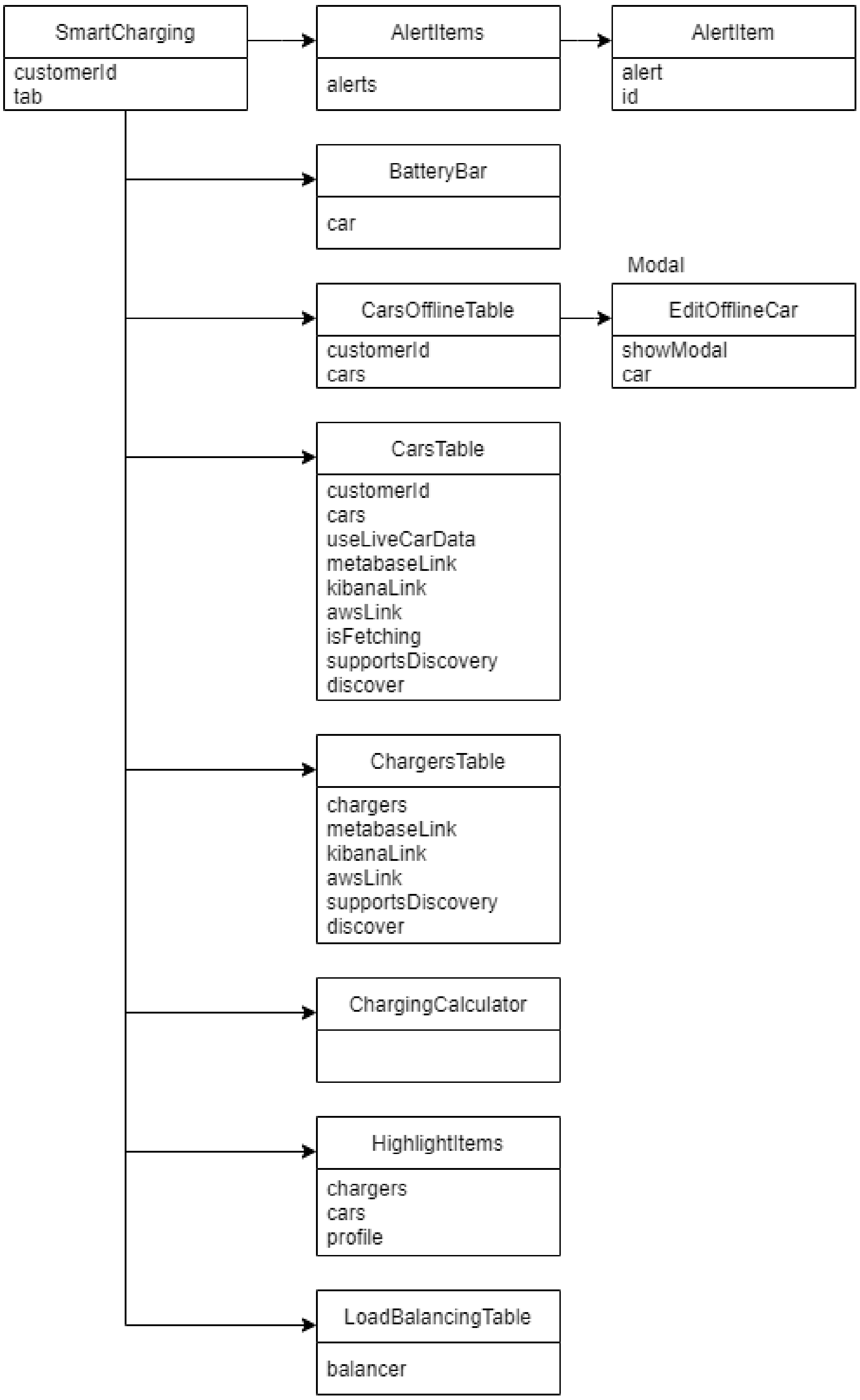


Figure 4.1 Smart Charging View structure.

5 DOMAIN MODEL

The domain model in figure 5.1 shows the domain of the customer. Each customer has at least one home with zero to many devices. These devices can be cars, chargers, and load balancers. There are multiple other types of devices that have been excluded from this model because they are not relevant to the development of the smart charging view.

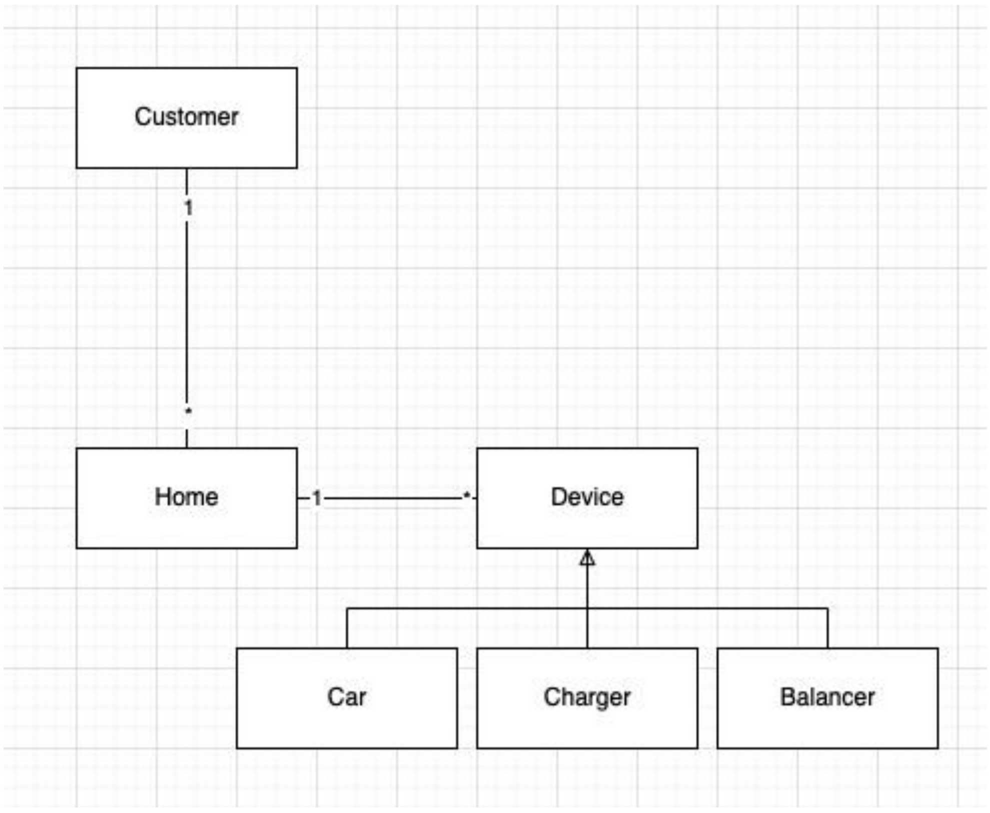


Figure 5.1 Domain model of a Customer object.

6 SECURITY

The smart charging view is a part of Varys and is therefore protected in the same way as Varys is. To be able to use Varys, a user must be both authenticated and authorized. The purpose of authentication is to confirm that users are who they claim to be. Once this has been confirmed, authorization is used to grant the user permission to access different levels of information and perform specific functions, based on the rules established for different user types (SailPoint, 2021).

Varys is an internal tool in Tibber, and it is only used by a subset of employees of Tibber. If you do not have a user with the correct credentials, gaining access to the system is quite difficult. First, a user must log in with their Tibber-account. Two-factor authentication is also obligatory, usually with an Authentication app on a personal mobile device. To use Varys and the smart charging view, the user must have the authorization to do so i.e., they must have the correct access credentials. An admin has authority to give other users access. This system was already in place before this project started and it was not deemed necessary that the team knows the inner workings of the security system.

To prevent unauthorized access to the system and avoid exploits and leaks, it is essential to adhere to best practices, such as storing passwords in a hashed and salted format, maintaining up-to-date libraries, etc. Thus, you avoid common security issues like rainbow tables, SQL injection and cross-site scripting.

7 INSTALLATION AND EXECUTION

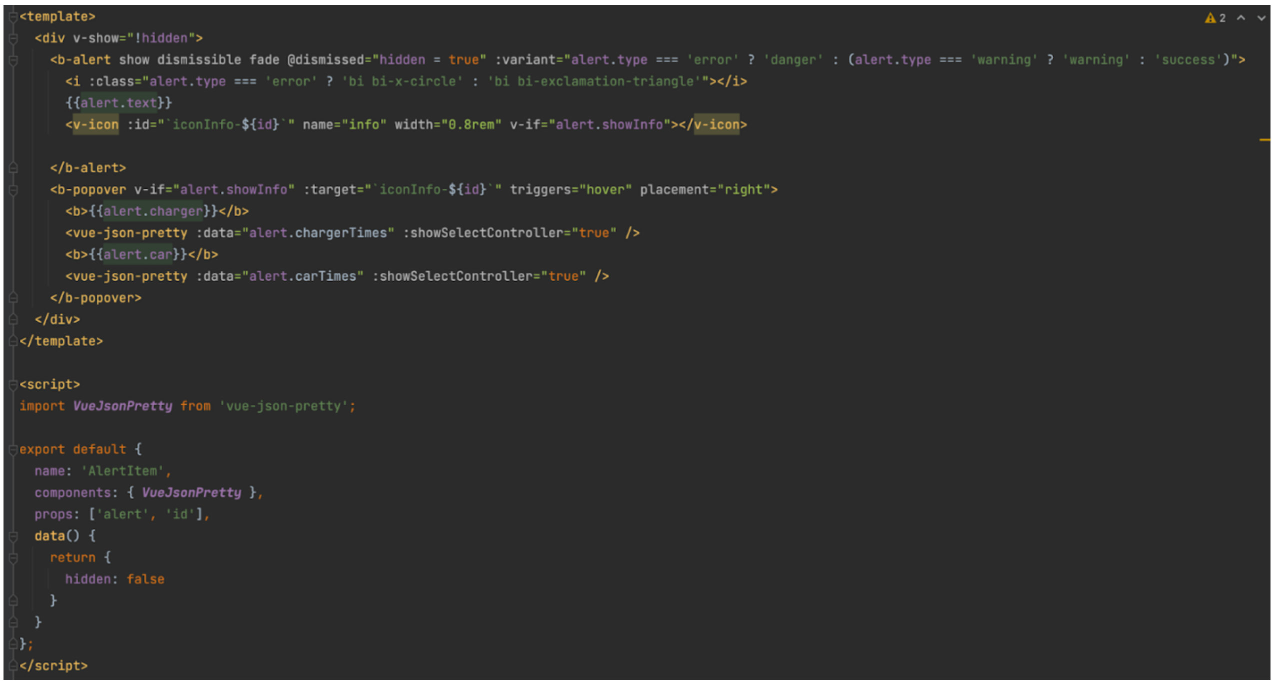
Docker is utilized to run test databases and proxies in order to access APIs from other running microservices, both for test and live data.

Yarn is used to install, operate, and test (via jest) the frontend and backend of Varys on a local machine. After fetching the project from GitHub, some secrets must be added to the product prior to running yarn install, which uses package.json to determine which libraries must be downloaded and then installs them.

Otherwise, the product is a part of a bigger web application that is constantly online and accessible in a web browser. Therefore, as long as the code is pushed to the production environment, it will always be possible to access the Smart Charging view. There is no need to start a program or similar to use the application.

8 DOCUMENTATION AND SOURCE CODE

Components are written in such a way that they are as generic and reusable as possible, meaning they can easily be used elsewhere in the application as long as their required properties are supplied by its parent. Below are examples of the most basic components in the project and how they are used.



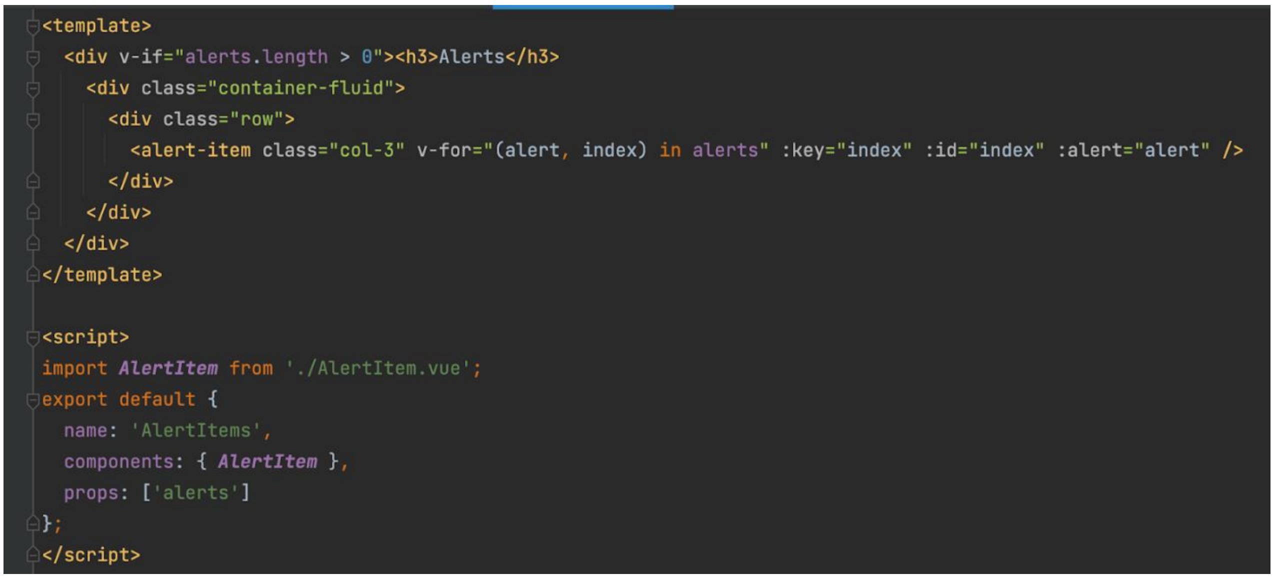
```
<template>
  <div v-show="!hidden">
    <b-alert show dismissible fade @dismissed="hidden = true" :variant="alert.type === 'error' ? 'danger' : (alert.type === 'warning' ? 'warning' : 'success')">
      <i :class="alert.type === 'error' ? 'bi bi-x-circle' : 'bi bi-exclamation-triangle'"></i>
      {{alert.text}}
      <v-icon :id="`iconInfo-${id}`" name="info" width="0.8rem" v-if="alert.showInfo"></v-icon>

    </b-alert>
    <b-popover v-if="alert.showInfo" :target="`iconInfo-${id}`" triggers="hover" placement="right">
      <b>{{alert.charger}}</b>
      <vue-json-pretty :data="alert.chargerTimes" :showSelectController="true" />
      <b>{{alert.car}}</b>
      <vue-json-pretty :data="alert.carTimes" :showSelectController="true" />
    </b-popover>
  </div>
</template>

<script>
import VueJsonPretty from 'vue-json-pretty';

export default {
  name: 'AlertItem',
  components: { VueJsonPretty },
  props: ['alert', 'id'],
  data() {
    return {
      hidden: false
    }
  }
};
</script>
```

Figure 8.1 The component used in AlertItems.vue.



```
<template>
  <div v-if="alerts.length > 0"><h3>Alerts</h3>
    <div class="container-fluid">
      <div class="row">
        <alert-item class="col-3" v-for="(alert, index) in alerts" :key="index" :id="index" :alert="alert" />
      </div>
    </div>
  </div>
</template>

<script>
import AlertItem from './AlertItem.vue';
export default {
  name: 'AlertItems',
  components: { AlertItem },
  props: ['alerts']
};
</script>
```

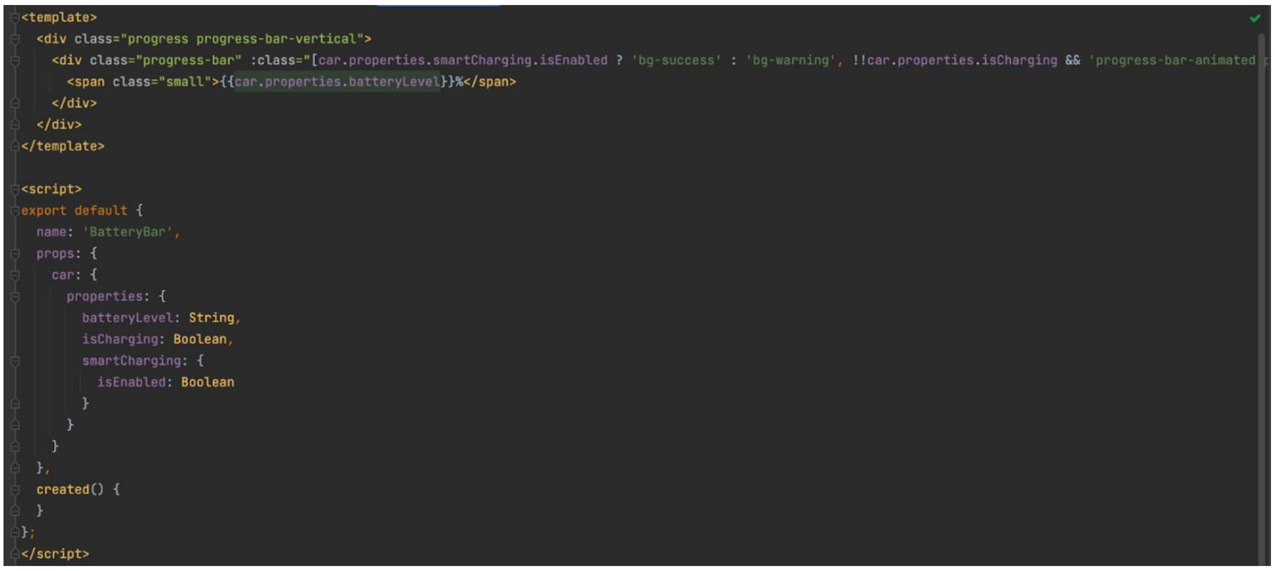
Figure 8.2 Usage of the AlertItem-component in index.vue.

Alerts (Green: “Good”, Orange: “Warning”, Red: “Error”):
The alert messages serve as a detailed summary; they also assist the support user with quick troubleshooting by automatically generating messages based on available devices and their

configuration state. Customer support agents can hover over the *‘i’-icon* to obtain additional information about the alert.

AlertItems-components are used in index.vue (see figure 8.2), the main file for the smart charging view. It is referred to with an <alert-item> tag. The required properties are passed on with dynamic properties, here named ‘:id’ and ‘:alert’.

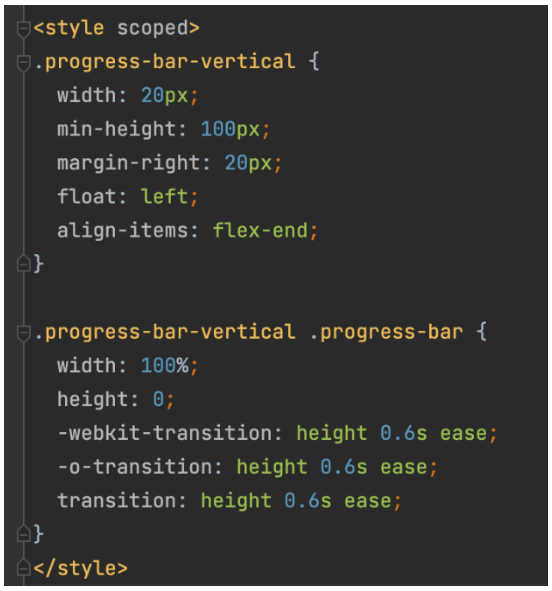
Figure 8.3 shows the BatteryBar.vue-file. It uses scoped CSS (see figure 8.4) to shift the default progress bar layout to a vertical one. This is used for displaying the current state of the car and whether smart charging is enabled. It will also be animated to indicate that a car is currently charging.



```
<template>
  <div class="progress progress-bar-vertical">
    <div class="progress-bar" :class="[car.properties.smartCharging.isEnabled ? 'bg-success' : 'bg-warning', !!car.properties.isCharging && 'progress-bar-animated
      <span class="small">{{car.properties.batteryLevel}}%</span>
    </div>
  </div>
</template>

<script>
export default {
  name: 'BatteryBar',
  props: {
    car: {
      properties: {
        batteryLevel: String,
        isCharging: Boolean,
        smartCharging: {
          isEnabled: Boolean
        }
      }
    }
  },
  created() {
  }
};
</script>
```

Figure 8.3 BatteryBar.vue.



```
<style scoped>
.progress-bar-vertical {
  width: 20px;
  min-height: 100px;
  margin-right: 20px;
  float: left;
  align-items: flex-end;
}

.progress-bar-vertical .progress-bar {
  width: 100%;
  height: 0;
  -webkit-transition: height 0.6s ease;
  -o-transition: height 0.6s ease;
  transition: height 0.6s ease;
}
</style>
```

Figure 8.4 "Scoped" CSS for BatteryBar.vue.

9 CONTINUOUS INTEGRATION AND TESTING

CircleCI includes testing as part of the build process. Typically, only builds that have successfully completed the development flow are deployable in both the development and production environments.

The popular Jest framework is used for unit testing. Important components include security and authority, as well as any essential functionality. For instance, only particular policy groups should have access to specific information or areas.

Vue Test Utils simplify the testing of Vue components. It enables testing of everything from View props and methods to properly rendered text.

Testing is an important part of CD/CI because it reduces the risk of critical errors and bugs and makes the process more efficient by reducing the time spent manually reading through code. It will not allow merging and will stop builds with failing tests. Therefore, test coverage should be sufficient but not overdone as writing tests is a time-consuming process.

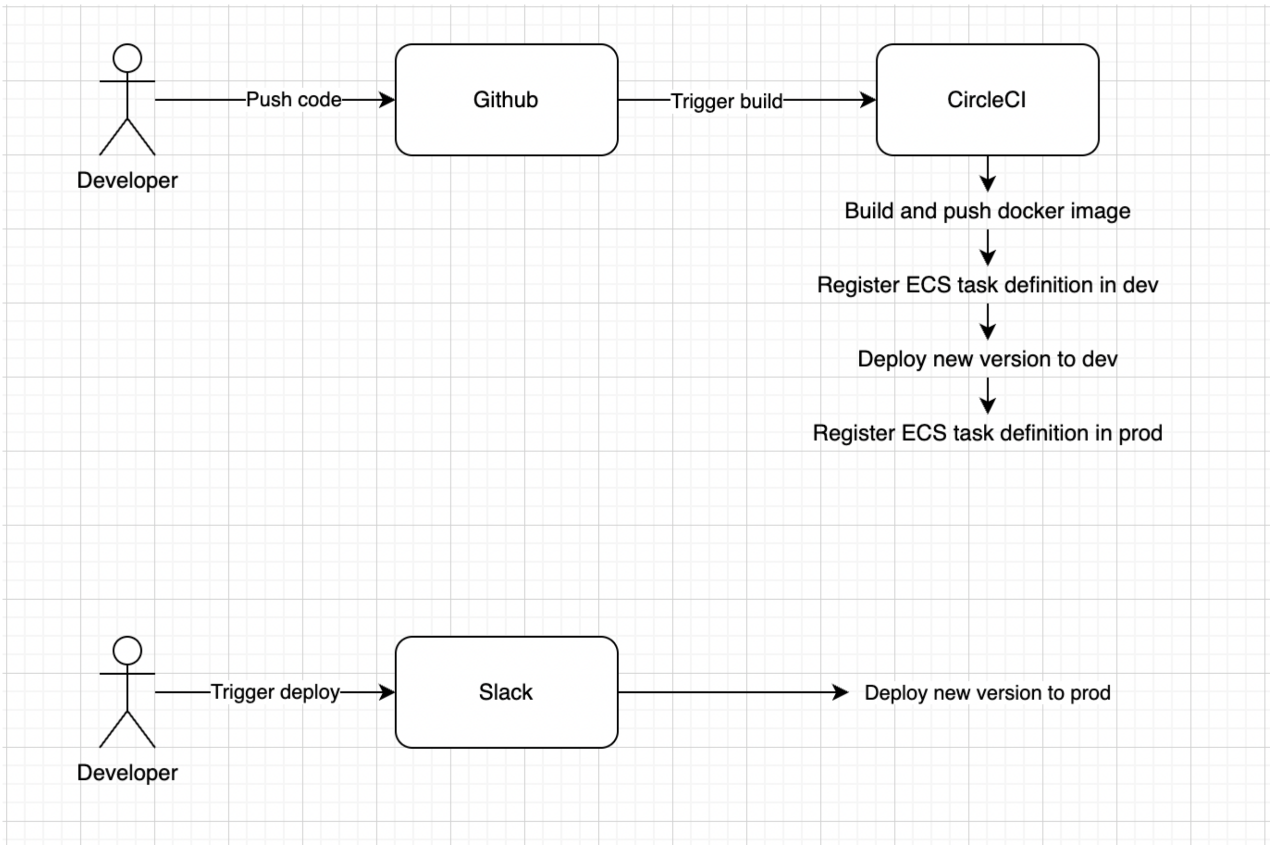

Developer
Push code
Github
Trigger build
CircleCI
Build and push docker image
Register ECS task definition in dev
Deploy new version to dev
Register ECS task definition in prod
Developer
Trigger deploy
Slack
Deploy new version to prod

Figure 9.1 Development and deployment flow to production.

10 REFERENCES

Docker (n.d.) Developers bring their ideas to life with Docker. Available at: https://www.docker.com/why-docker/ (Retrieved May 16, 2022).

Microsoft (2021) SQL Injection. Available at: https://docs.microsoft.com/en-us/sql/relational-databases/security/sql-injection?view=sql-server-ver15 (Retrieved Apr 12, 2022).

NewRelic (n.d.) Welcome to New Relic. Available at: https://docs.newrelic.com/docs/new-relic-solutions/get-started/intro-new-relic (Retrieved Mar 2, 2022).

Sailpoint (2021) What Is the Difference between Authentication and Authorization? Available at: https://www.sailpoint.com/identity-library/difference-between-authentication-and-authorization/ (Retrieved Apr 12, 2022).