VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

HUAN DO

EXPENSE TRACKING APPLICATION

School of Technology
2023

VAASAN AMMATTIKORKEAKOULU
University of Applied Sciences

ABSTRACT

Author	Huan Do
Title	Expense Tracking Application
Year	2023
Language	English
Pages	37
Supervisor	Anna Kaisa Saari

It came from the idea of helping a friend to keep track of expenses and from that they can manage income and outcome. She wanted to have an application which can insert shopping receipts from stores. And from that, being able to track monthly expenses, therefore make better spending decisions.

The objective of this thesis work was to build and deploy a fully responsive application using modern UI with Angular Framework. The major reason for choosing Angular is because this Framework is known to be one of the most widely used and has a sizable library free to use and easy to access to.

The project was fully tested and proved to be working as required. It also achieved the aim of being responsive to support also mobile users.

Keywords Angular, Expense, Tracker

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

There has been a rapid change in the technology industry. One of the remarkable changes is many ways to create tracking products which help to make our life easier.

Out of many new programming languages, JavaScript with Angular Framework stands out to be one of the most efficient ways to create such applications.

1.1 Study Case

The original purpose for creating the site came from the idea to help a friend with tracking, which allows her to track daily expenses such as what products are bought or where the money is spent. This tracking application allows expense tracking and gives users statistics about their expenses.

The front-end of the application is written in JavaScript using the Angular framework, utilizing HTML and CSS (Cascading Style Sheets). In addition to these PrimeNG library for UI components was used. At the back end of the application the following technologies were used: the Expressjs framework, JavaScript, also bcrypt (for password hashing and encrypting), JSON Web Token (for authorization and authentication) were used along with Dotenv.

1.2 Scope and objectives

The main objective of this thesis project is to make expense tracking process easier. The goal for the application interface is to have the most crucial information available: balance and expenses.

2 REQUIREMENTS

2.1 Description

The basic idea for the Expense tracking application is to balance the user’s income against expenses. With the application, the user can input their shopping information, such as store information, date for shopping, total sum, receipt or other images and categorize the information for further analysis and tracking.

2.2 Use Cases

The use-case diagram, which describes possible interactions between users and the system of the application is shown in Figure 1. The application has the following functionalities:

1. Log in and log out, where the user can create a new account, specify user information and log out of the application
2. Add new transactions, where users can add their income and receipts from shopping
3. View and edit existing transactions
4. Categories to categorize user expenses. Users can add new categories or edit existing ones.
5. Statistics that include a line chart and a pie chart to show the difference between incomes and outcomes that changes over months and expenses divided by categories

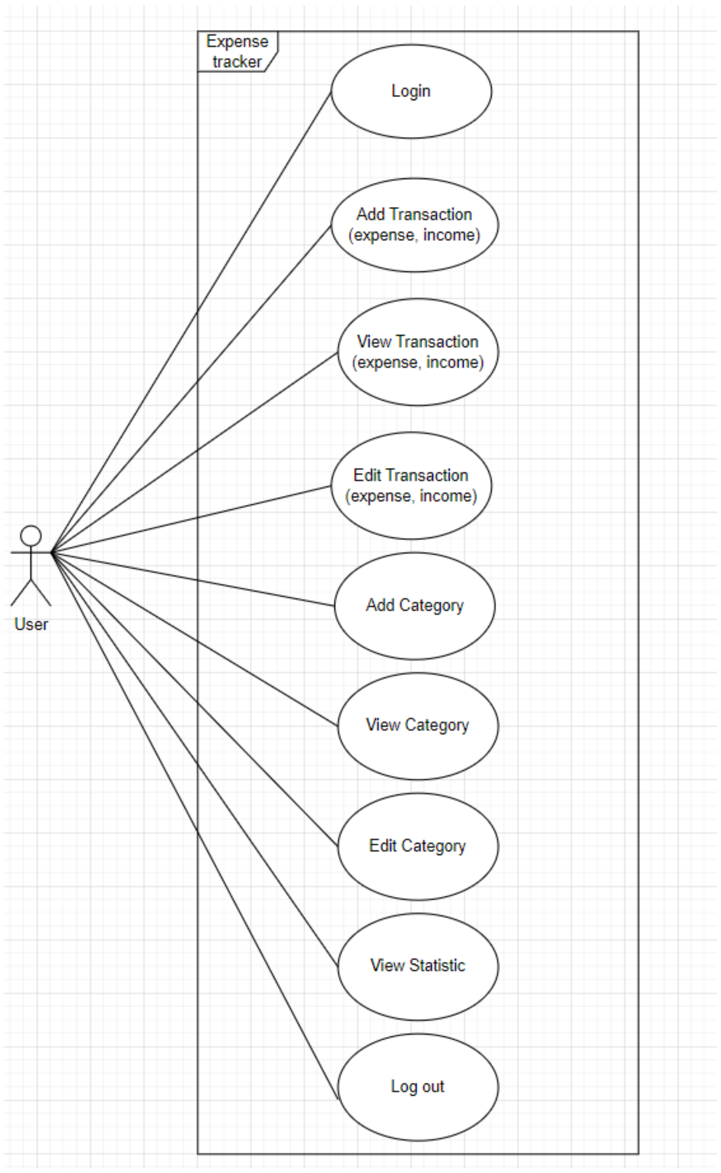

Expense tracker
Login
Add Transaction (expense, income)
View Transaction (expense, income)
Edit Transaction (expense, income)
Add Category
View Category
Edit Category
View Statistic
Log out
User

Figure 1. Use-case Diagram

3 RELEVANT TECHNOLOGIES

3.1 JavaScript, HTML and CSS

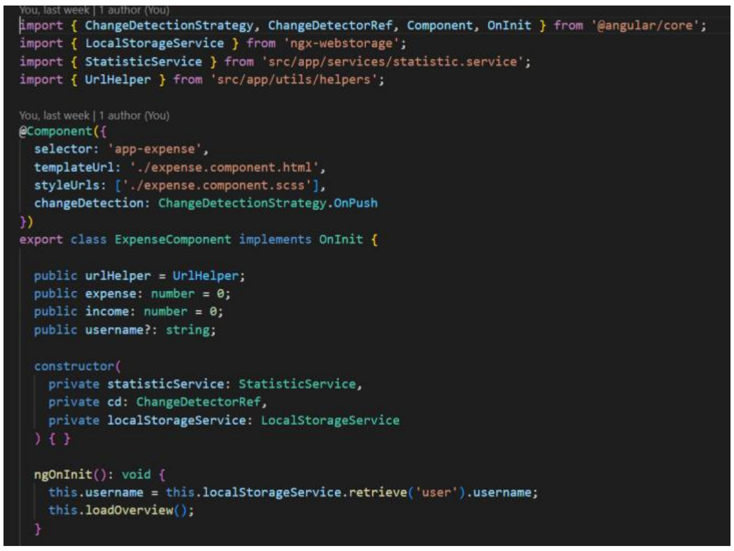
The application was written in JavaScript using the Angular framework. JavaScript, often abbreviated as JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. According to Wikipedia, as of 2022, 98% of the websites use JavaScript on the client side for webpage behavior.

The rise in the use of mobile devices has led many a business owner to consider if the business requires both a website and a mobile application. With JavaScript, the project also utilizes HTML and CSS. CSS is used to style an HTML document. It describes how HTML elements should be displayed and how applications work smoothly in different devices. /1//2/

3.2 Angular Framework

Angular (also referred to as “Angular 2+”) is a TypeScript-based /3/, free and open-source web application framework, whose development is led by the Angular Team and by a community of individuals. Angular is a Single Page application Framework which is used for creating Web Applications fast.

An example of a minimal Angular component is shown in Figure 2. The class is marked by decorator @Component, which makes it one component. Once component can implement one life cycle, for example OnInit is used.

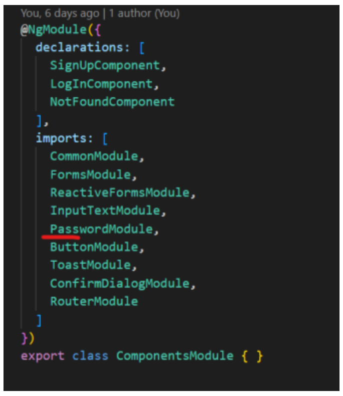


```
You, last week | 1 author (You)
import { ChangeDetectionStrategy, ChangeDetectorRef, Component, OnInit } from '@angular/core';
import { LocalStorageService } from 'ngx-webstorage';
import { StatisticService } from 'src/app/services/statistic.service';
import { UrlHelper } from 'src/app/utils/helpers';

You, last week | 1 author (You)
@Component({
  selector: 'app-expense',
  templateUrl: './expense.component.html',
  styleUrls: ['./expense.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class ExpenseComponent implements OnInit {

  public urlHelper = UrlHelper;
  public expense: number = 0;
  public income: number = 0;
  public username?: string;

  constructor(
    private statisticService: StatisticService,
    private cd: ChangeDetectorRef,
    private localStorageService: LocalStorageService
  ) { }

  ngOnInit(): void {
    this.username = this.localStorageService.retrieve('user').username;
    this.loadOverview();
  }
```

Figure 2. Angular components

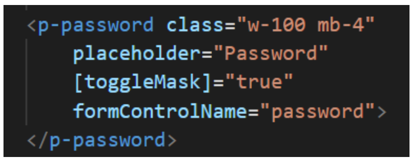
3.3 UI library PrimeNG

PrimeNG is a collection of rich UI components for Angular. All widgets are open and free to use under the MIT License. The PrimeNG UI library makes front-end development fast and efficient. PrimeNG is written in TypeScript, and it does not contain any third-party dependencies. /3/



```
You, 6 days ago | 1 author (You)
@NgModule({
  declarations: [
    SignUpComponent,
    LogInComponent,
    NotFoundComponent
  ],
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    InputTextModule,
    PasswordModule,
    ButtonModule,
    ToastModule,
    ConfirmDialogModule,
    RouterModule
  ]
})
export class ComponentsModule { }
```

Figure 3. PasswordModule is imported from UI library



```
<p-password class="w-100 mb-4"
    placeholder="Password"
    [toggleMask]="true"
    formControlName="password">
</p-password>
```

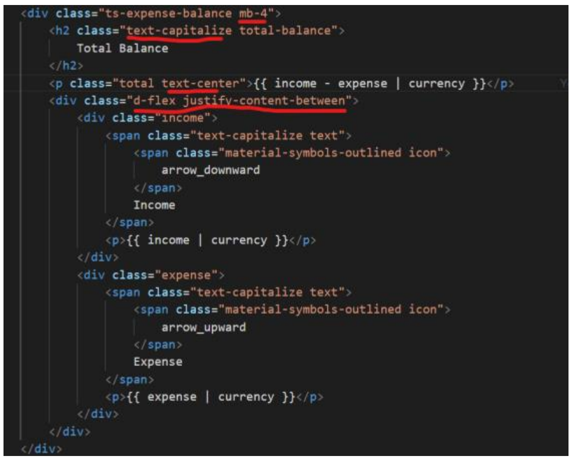
Figure 4. Utilizing properties in “p-password”

The Password component is defined and exported in PasswordModule. To be able to use the p-password component, passwordModule needs to be imported, as shown in Figure 3. As shown in Figure 4, properties such as toggleMask are used from the p-password. ToggleMask is to show an eye-like icon that displays password as plain text; the placeholder is for advisory information to display on input, see Figure 4

3.4 Bootstrap

Bootstrap /4/ is the most popular CSS Framework for developing responsive and mobile-first websites. Bootstrap 5 is the newest version Bootstrap. Bootstrap is used in this project because of its conveniences, classes are well designed for responsive purpose.

Bootstrap contains predefined classes and by using them developers do not need to write the CSS by themselves. For example, as shown in Figure 5, Bootstrap class .mb-4 replaces CSS {margin-bottom: $spacer*1.5}. In the second <div/>, .d-flex replaces CSS {display: flex}; .justify-content-between replaces for CSS {justify-content: between}.

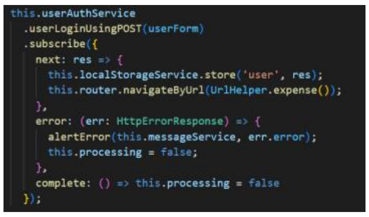


```
<div class="ts-expense-balance mb-4">
    <h2 class="text-capitalize total-balance">
        Total Balance
    </h2>
    <p class="total text-center">{{ income - expense | currency }}</p>
    <div class="d-flex justify-content-between">
        <div class="income">
            <span class="text-capitalize text">
                <span class="material-symbols-outlined icon">
                    arrow_downward
                </span>
                Income
            </span>
            <p>{{ income | currency }}</p>
        </div>
        <div class="expense">
            <span class="text-capitalize text">
                <span class="material-symbols-outlined icon">
                    arrow_upward
                </span>
                Expense
            </span>
            <p>{{ expense | currency }}</p>
        </div>
    </div>
</div>
```

Figure 5. Ultilizing Spacing in Bootstrap

3.5 RxJS

RXJS (Reactive Extensions Library for JavaScript) is a library for reactive programming. RXJS uses observables, which makes it easier to compose asynchronous or call-back-based code. It offers a powerful, functional approach for dealing with events and integration points into many frameworks. /6//20/



```
this.userAuthService
  .userLoginUsingPOST(userForm)
  .subscribe({
    next: res => {
      this.localStorageService.store('user', res);
      this.router.navigateByUrl(UrlHelper.expense());
    },
    error: (err: HttpErrorResponse) => {
      alertError(this.messageService, err.error);
      this.processing = false;
    },
    complete: () => this.processing = false
  });
```

Figure 6. The use of RXJS

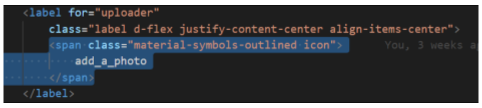
Figure 6 shows how RXJS is used to call Login-API. When the API returns data, the observer can receive three events: “next”, “error”, “complete”. The “Next” event is triggered when the observer receives a successful value, and the “Error” event is triggered when the observer receives an error value. “complete” is called after the whole stream is completed.

If the username and password are incorrect, the application returns error “User not found” to the user. If the username is correct but a wrong password was given, the application returns “Invalid password” to the user.

3.6 Google Icons

Google Icons is an icon set that consolidates over 2,891 glyphs in a single font file with a wide range of design variants. The symbols are available in three styles and four adjustable variable font styles (fill, weight, grade, and optical size). /7/

In this project, “add a photo” icon was used. In order to have “add a photo” icon, tag <span/> must have class “material-symbols-outline” with content add_a_photo, as shown in Figure 7.

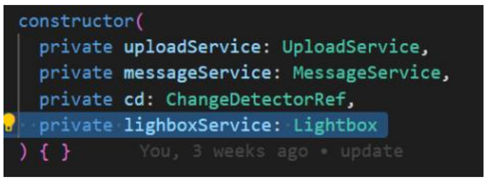


```
<label for="uploader"
    class="label d-flex justify-content-center align-items-center">
    <span class="material-symbols-outlined icon">        You, 3 weeks a
        add_a_photo
    </span>
</label>
```

Figure 7. Adding the target icon in <span/> tag

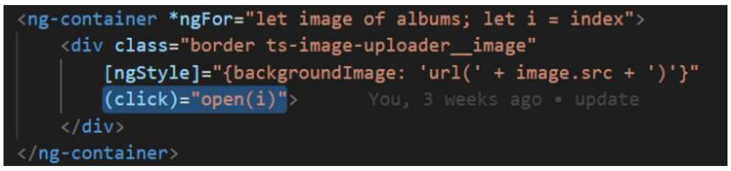
3.7 LightBox

Lightbox is a small JavaScript library that is used to overlay images on top of the current page. It is an easy to set up and it works on all modern browsers. /8/ LightBox provides services, which enable services injection into components via constructor, as shown in Figure 8. In this template, (click) event will be listened to call function open() and this function will have an index parameter (Figure 9). To prepare data for LightBox, an array <IAlbum> is predefined (Figure 10); in Figure 11, LightBoxservices provide open() function. This function receives 3 parameters: Album, index, config to pop up the photo.



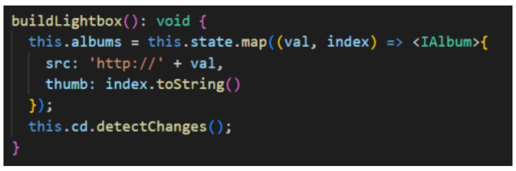
```
constructor(
  private uploadService: UploadService,
  private messageService: MessageService,
  private cd: ChangeDetectorRef,
  private lighboxService: Lightbox
) { }        You, 3 weeks ago • update
```

Figure 8. Inject Lightbox



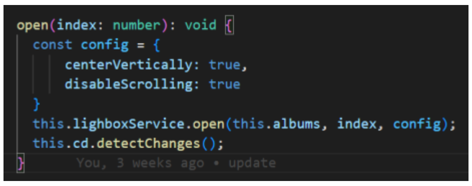
```
<ng-container *ngFor="let image of albums; let i = index">
    <div class="border ts-image-uploader__image"
        [ngStyle]="{backgroundImage: 'url(' + image.src + ')'}"
        (click)="open(i)">        You, 3 weeks ago • update
    </div>
</ng-container>
```

Figure 9. (click) event & open(i) function



```
buildLightbox(): void {
  this.albums = this.state.map((val, index) => <IAlbum>{
    src: 'http://' + val,
    thumb: index.toString()
  });
  this.cd.detectChanges();
}
```

Figure 10. Array <IAlbum> is predefined



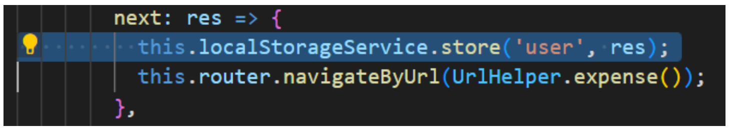
```
open(index: number): void {
  const config = {
      centerVertically: true,
      disableScrolling: true
  }
  this.lighboxService.open(this.albums, index, config);
  this.cd.detectChanges();
}       You, 3 weeks ago • update
```

Figure 11. function open() takes 3 parameters

3.8 Ngx-webstorage

The ngx-webstorage library provides an easy-to-use service to manage the web storages of browser from an Angular application. It provides also two decorators to synchronize the component attributes and the web storages /9/

In the project, LocalStorageservice was injected to components via a constructor. The library provides a function called store() that takes two parameters, key and its value. This combination is then saved into localStorage, as shown in Figure 12.



```
next: res => {
  this.localStorageService.store('user', res);
  this.router.navigateByUrl(UrlHelper.expense());
},
```

Figure 12. LocalStorageServices injection to components

4 DATABASE & GUI DESIGN

4.1 Database Diagram

This chapter describes the structure of the database and relations between database objects. The needed objects are User, Category and Expense, as shown in Figure 13. For user and category, the relation is one to many, meaning that one user can have many categories. For category and expense, the relation is many to many, meaning that many categories can have many expenses.

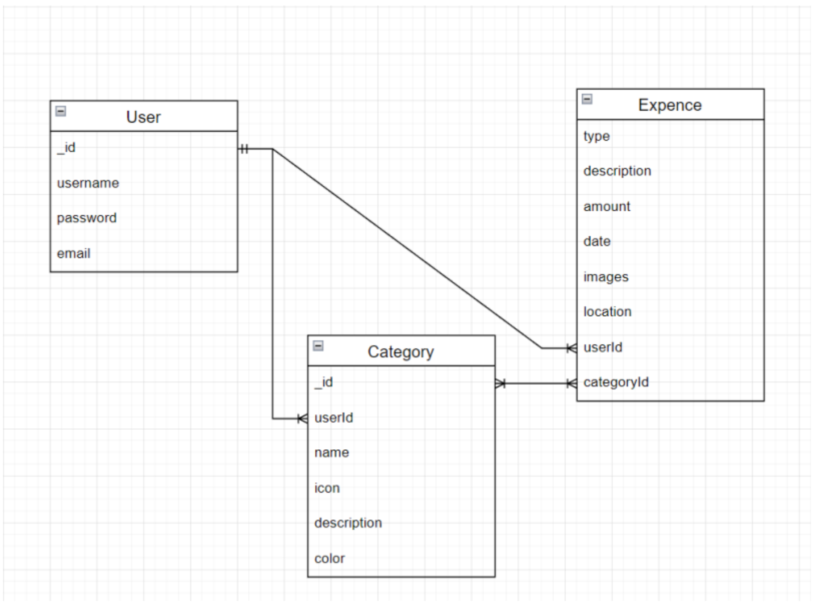


Figure 13. Relations between tables

4.2 Process Flow

This chapter includes a process flow chart which describes how the application works. The process flow is shown in Figure 14. Before starting the application, the user must register to the system. The user can create an account in the Sign in view. Once the account is created, the user is directed back to the Login page.

On the Login page, a wrong password directs the user back to the Login with an error notification, the right password directs to the home page. At the Home page, the user can either go to Expense handling or log out from the application. In case of Expense which not been added yet (does not exist), the user can create new

Expense, where a database for Expense is needed. A cycle of Expense is as follows: no category exists >create a new category (database needed), category exists, then the user is able to edit the category (in category database created). On the other hand, if Expense already exists, then the user can go straight to edit expense (interact with database) on the Homepage.

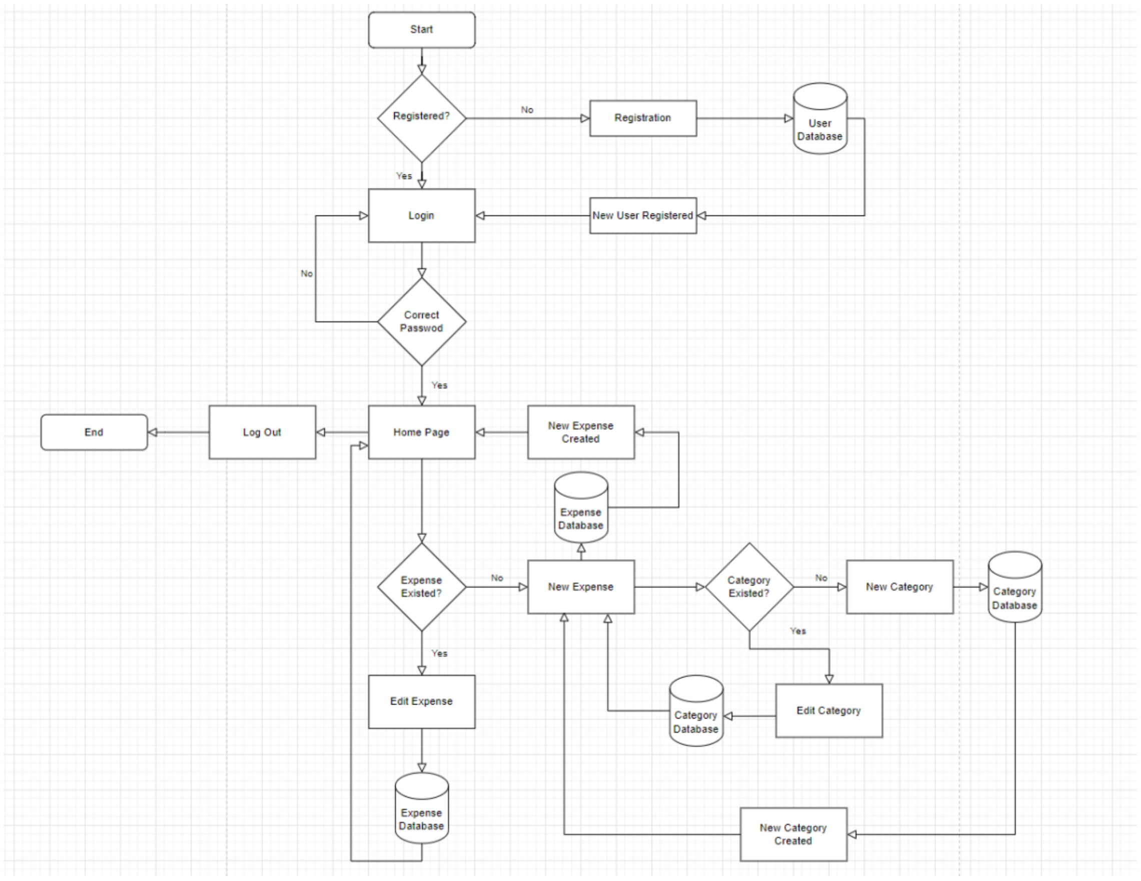
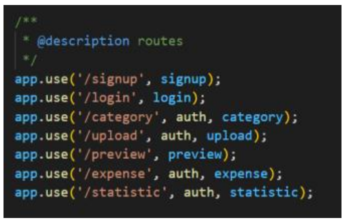


Figure 14. Process Flow Diagram

4.3 Other technologies

4.3.1 Framework ExpressJS

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and applications. It is used to build a single page, multipage, and hybrid web application. /5/ As shown in Figure 15, the application can receive requests from clients using different paths. These paths are routed to corresponding routes.



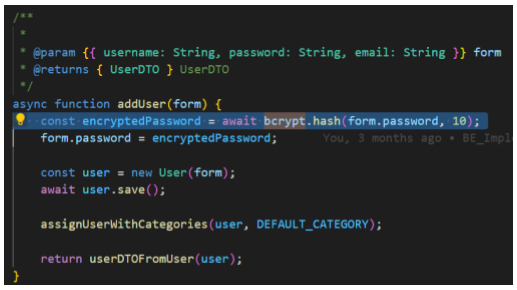
```
/**
 * @description routes
 */
app.use('/signup', signup);
app.use('/login', login);
app.use('/category', auth, category);
app.use('/upload', auth, upload);
app.use('/preview', preview);
app.use('/expense', auth, expense);
app.use('/statistic', auth, statistic);
```

Figure 15. Bootstrap the app by PORT

4.3.2 Bcrypt

Bcrypt is a password-hashing function designed by Niels Provos and David Mazieres. The bcrypt function is the default password hash algorithm. There are implementations of bcrypt in C, C++, Go, Java, JavaScript and other languages /10/

An example of the password hashing process can be seen in Figure 16. Bcrypt has function hash() that helps to encrypt passwords. Then it changes the user's password into an encrypted password using the hash() function bcrypt provides. The encrypted password of the corresponding user is saved to the database.



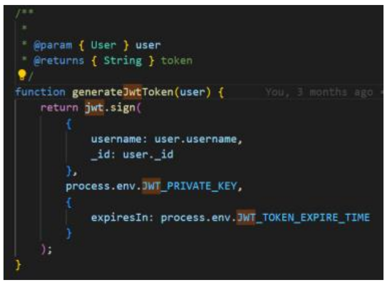
```
/**
 *
 * @param {{ username: String, password: String, email: String }} form
 * @returns { UserDTO } UserDTO
 */
async function addUser(form) {
    const encryptedPassword = await bcrypt.hash(form.password, 10);
    form.password = encryptedPassword;        You, 3 months ago • BE_Impl

    const user = new User(form);
    await user.save();

    assignUserWithCategories(user, DEFAULT_CATEGORY);

    return userDTOFromUser(user);
}
```

Figure 16. Hashing passwords

4.3.3 JSON Web Token

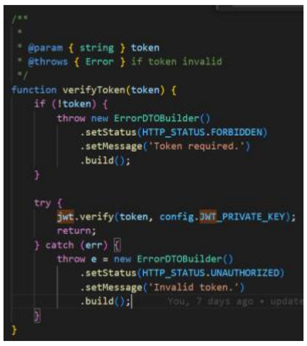
JSON Web Token (or JWTs) is an open, industry standard RFC 7519 method for representing claims securely between two parties. As a JSON object, information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (HMAC algorithm) or a public/private key pair using RSA or ESDSA /11/

In this thesis project, JWT was used for authorization reasons. Once the user logs in, they are allowed to access routes, services, and resources that are permitted with that token. Assigned 'jsonwebtoken' as variable jwt (Figure 17), a token is generated from the user information: id, username with private key and expire time. In Figure 18, JWT provides a verify() function that takes 2 parameters: token (needs to be verified) and private key and return true if the token is valid, when the token is invalid, the function will throw an error.



```
/**
 *
 * @param { User } user
 * @returns { String } token
 */
function generateJwtToken(user) {        You, 3 months ago
    return jwt.sign(
        {
            username: user.username,
            _id: user._id
        },
        process.env.JWT_PRIVATE_KEY,
        {
            expiresIn: process.env.JWT_TOKEN_EXPIRE_TIME
        }
    );
}
```

Figure 17. Generate token function

, 

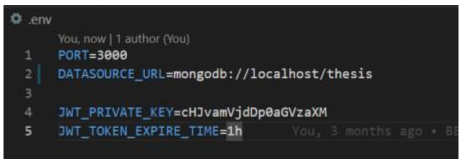
```
/**
 *
 * @param { string } token
 * @throws { Error } if token invalid
 */
function verifyToken(token) {
    if (!token) {
        throw new ErrorDTOBuilder()
            .setStatus(HTTP_STATUS.FORBIDDEN)
            .setMessage('Token required.')
            .build();
    }

    try {
        jwt.verify(token, config.JWT_PRIVATE_KEY);
        return;
    } catch (err) {
        throw e = new ErrorDTOBuilder()
            .setStatus(HTTP_STATUS.UNAUTHORIZED)
            .setMessage('Invalid token.')
            .build();        You, 7 days ago • update
    }
}
```

Figure 18. Verify token

4.3.4 Dotenv

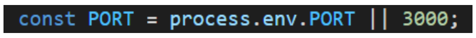
Dotevn is a zero-dependency module that loads environment variables from a .env file into process.env. The Dotenv package is a great way to keep passwords, API keys, and other sensitive data out of codes. It allows creating environment variables in a .env file /12/

In the project, .env file was created and variables were defined with syntax: variable_name = value, as shown in Figure 19. These variables can be accessed using syntax process.env.variable_name, as shown in Figure 20



```
.env
You, now | 1 author (You)
PORT=3000
DATASOURCE_URL=mongodb://localhost/thesis

JWT_PRIVATE_KEY=cHJvamVjdDp0aGVzaXM
JWT_TOKEN_EXPIRE_TIME=1h        You, 3 months ago • BE
```

Figure 19. Define environment variable

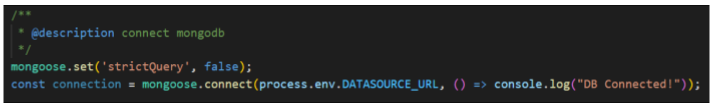


```
const PORT = process.env.PORT || 3000;
```

Figure 20. Using PORT variable

4.3.5 Mongoose

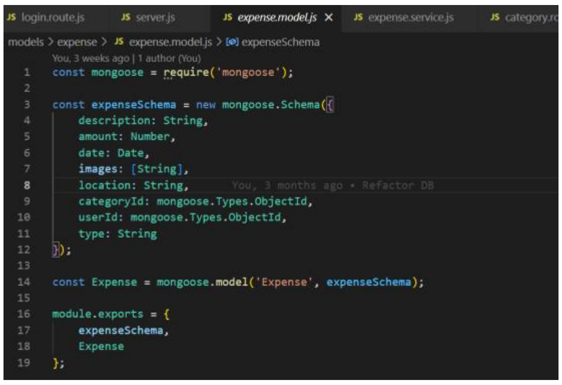
Mongoose is a MongoDB object modeling tool, designed to work in an asynchronous environment /14/. Mongoose 7.0.0 version was used in this project. First, Mongoose is required, then it was used to connect database via connect() function (Figure 21),



```
/**
 * @description connect mongodb
 */
mongoose.set('strictQuery', false);
const connection = mongoose.connect(process.env.DATASOURCE_URL, () => console.log("DB Connected!"));
```

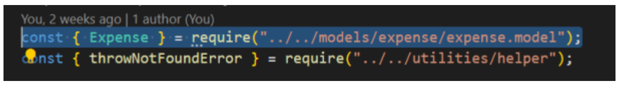
Figure 21. Connect to database

A schema can be defined for the database (Figure 22). The schema helps to interact with the database through the functions, as shown in Figure 23 and 24


login.route.js
server.js
expense.model.js
expense.service.js
category.ro
models > expense > expense.model.js > expenseSchema
You, 3 weeks ago | 1 author (You)
You, 3 months ago • Refactor DB

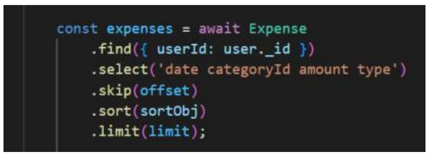
```
const mongoose = require('mongoose');

const expenseSchema = new mongoose.Schema({
    description: String,
    amount: Number,
    date: Date,
    images: [String],
    location: String,
    categoryId: mongoose.Types.ObjectId,
    userId: mongoose.Types.ObjectId,
    type: String
});

const Expense = mongoose.model('Expense', expenseSchema);

module.exports = {
    expenseSchema,
    Expense
};
```

Figure 22. Define a Schema



```
const { Expense } = require("../../models/expense/expense.model");
const { throwNotFoundError } = require("../../utilities/helper");
```

Figure 23.Import Schema



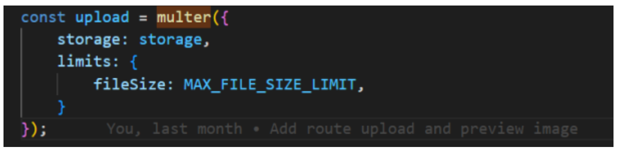
```
const expenses = await Expense
    .find({ userId: user._id })
    .select('date categoryId amount type')
    .skip(offset)
    .sort(sortObj)
    .limit(limit);
```

Figure 24. Using Schema to fetch data

4.3.6 Multer

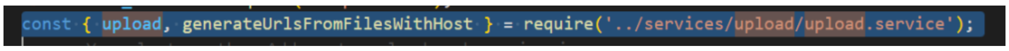
Multer is a node.js middleware for handling multipart/form-data. It is primarily used for uploading files purpose and is written on top of busboy for maximum efficiency. Multer will not process any form which is not multipart (multipart/form-data) /15/

Multer adds a body object and a file (or files) object to the request object. The body object contains the values of the text fields of the form, the file of files object contains the files uploaded via the form. First, 'multer' is required, then a middleware upload is defined with configs (see Figure 25), then the upload is imported into needed service, in Figure 26. Finally, this middleware is used to receive and store files to the database from the request, as can be seen in Figure 27



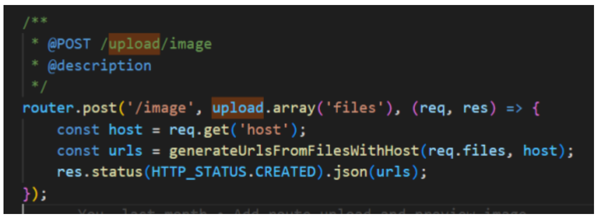
```
const upload = multer({
    storage: storage,
    limits: {
        fileSize: MAX_FILE_SIZE_LIMIT,
    }
});          You, last month • Add route upload and preview image
```

Figure 25. Defining the upload middleware with config



```
const { upload, generateUrlsFromFilesWithHost } = require('../services/upload/upload.service');
```

Figure 26. Importing the upload middleware

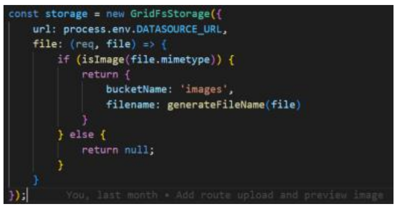


```
/**
 * @POST /upload/image
 * @description
 */
router.post('/image', upload.array('files'), (req, res) => {
    const host = req.get('host');
    const urls = generateUrlsFromFilesWithHost(req.files, host);
    res.status(HTTP_STATUS.CREATED).json(urls);
});
```

Figure 27. Using the upload middleware

4.3.7 GridFS

GridFS is a specification for storing and retrieving files that exceed the BSON - document size limit of 16 MB. Instead of storing a file in a single document, GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. In this thesis project, multer-grid-storage supports the software storing files into MongoDB. In Figure 28, a storage with configs (url & file) is then defined.



```
const storage = new GridFsStorage({
    url: process.env.DATASOURCE_URL,
    file: (req, file) => {
        if (isImage(file.mimetype)) {
            return {
                bucketName: 'images',
                filename: generateFileName(file)
            }
        } else {
            return null;
        }
    }
});          You, last month • Add route upload and preview image
```

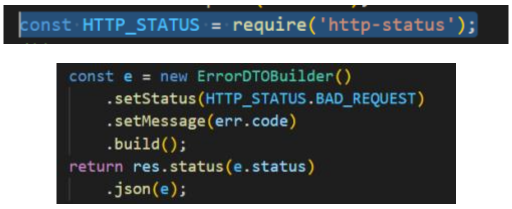
Figure 28. Define a storage to save photos

4.3.8 CORS

Cross-origin recourse sharing (CORS) is a mechanism which allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served /17/. In the project, CORS was used because the two applications (FE&BE) are deployed in 2 different servers.

4.3.9 Http-status

The HTTP Status code for node creates Utility for interacting with the HTTP status code. Once this module is required, one can call it with either an HTTP code or a message name. With an HTTP code, one can get the message name while with a message name one will get an HTTP code. /18/ In this project, instead of return status code “403”, HTTP_STATUS.BAD_REQUEST can be used when there is an error, see Figure 29.



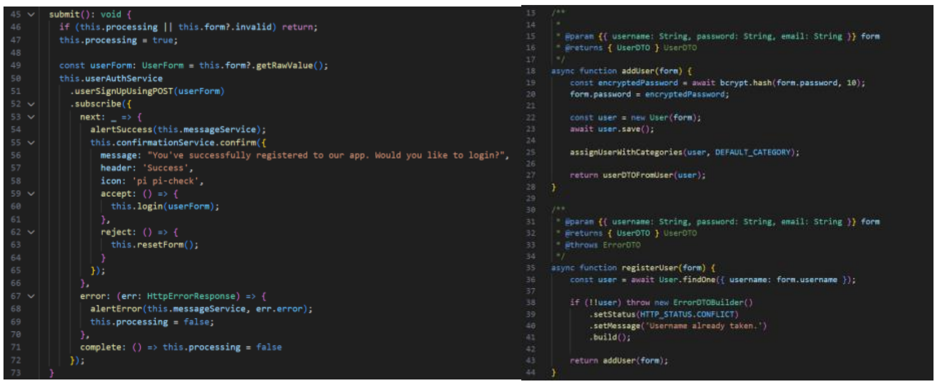
```
const HTTP_STATUS = require('http-status');
```

```
const e = new ErrorDTOBuilder()
    .setStatus(HTTP_STATUS.BAD_REQUEST)
    .setMessage(err.code)
    .build();
return res.status(e.status)
    .json(e);
```

Figure 29. Creating an error with status 'BAD_REQUEST'

5 IMPLEMENTATION, DEPLOYMENT & TESTING

5.1 Implementation

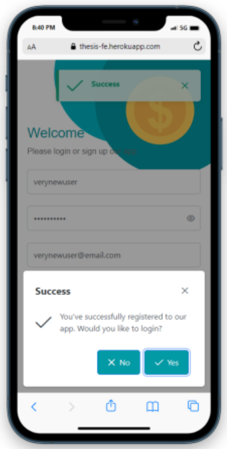


```
submit(): void {
  if (this.processing || this.form?.invalid) return;
  this.processing = true;

  const userForm: UserForm = this.form?.getRawValue();
  this.userAuthService
    .userSignUpUsingPOST(userForm)
    .subscribe({
      next: _ => {
        alertSuccess(this.messageService);
        this.confirmationService.confirm({
          message: "You've successfully registered to our app. Would you like to login?",
          header: 'Success',
          icon: 'pi pi-check',
          accept: () => {
            this.login(userForm);
          },
          reject: () => {
            this.resetForm();
          }
        });
      },
      error: (err: HttpErrorResponse) => {
        alertError(this.messageService, err.error);
        this.processing = false;
      },
      complete: () => this.processing = false
    });
}
```

```
/**
 *
 * @param {{ username: String, password: String, email: String }} form
 * @returns { UserDTO } UserDTO
 */
async function addUser(form) {
    const encryptedPassword = await bcrypt.hash(form.password, 10);
    form.password = encryptedPassword;

    const user = new User(form);
    await user.save();

    assignUserWithCategories(user, DEFAULT_CATEGORY);

    return userDTOFromUser(user);
}

/**
 * @param {{ username: String, password: String, email: String }} form
 * @returns { UserDTO } UserDTO
 * @throws ErrorDTO
 */
async function registerUser(form) {
    const user = await User.findOne({ username: form.username });

    if (!!user) throw new ErrorDTO Builder()
        .setStatus(HTTP_STATUS.CONFLICT)
        .setMessage('Username already taken.')
        .build();

    return addUser(form);
}
```

Figure 30. Sign Up

When the user accesses the application, they will be directed to the Login with path= "/login". The Sign-up page has a form (see Figure 30) that takes input from the client, and checks if that input is valid. If the input is valid, the front end sends the form data to the backend API. If the user is a new user, an account is created. When the user succeeded in creating a new account, the application will pop up, as in Figure 31. Another feature when accessing the url to the software application, when user gives wrong path of the url, the application will pop up an image with status "404" not found, as shown in figure 32.



verynewuser

••••••••••

verynewuser@email.com

Success

You've successfully registered to our app. Would you like to login?

Figure 31. Log In Success

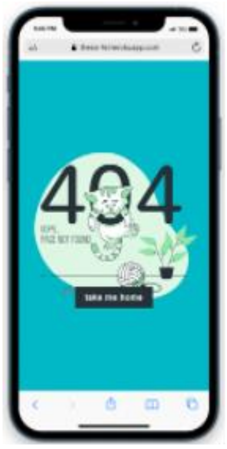
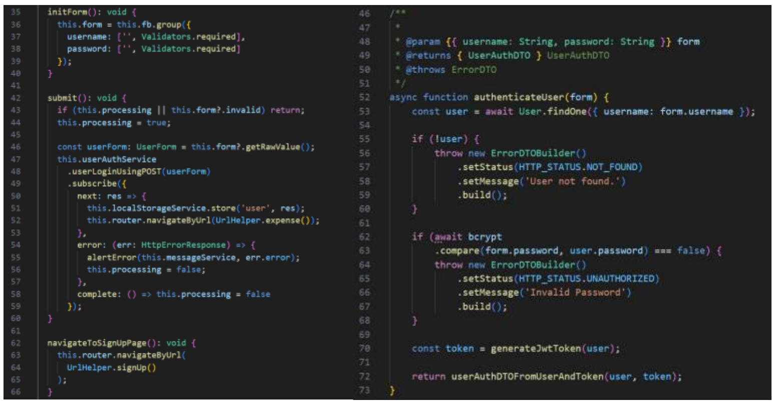


Figure 32. Wrong path with 404 not found status

In the component login.component.ts, a form variable type FormGroup, assigns two field inputs “username” and “password” via directive formConTrolName. The Login also check the form of the user input, if the input is valid, the form is sent to the backend. The backend takes over the form, checking the user information from the database, if it matches, then allows the user to log themselves in. Here, Validators were used for both fields, “username” & “password”, if any of the above fields does not have a value then the form is invalid, utilizing these properties to handle logic, for example not allowing the request to be sent. In the case the form is valid, HTTPClient is used to send one request to the backend with a body that includes the username and password. Meanwhile, the backend will verify the user and return the corresponding result, as seen in Figure 33.

If the user is valid, localStorageService is used to store the user information, including the token into LocalStorage. Tokens will be used to authenticate users in future requests. Below is the demo images of Sign up and Log in UI of the software, as shown in Figure 34.



```
initForm(): void {
  this.form = this.fb.group({
    username: ['', Validators.required],
    password: ['', Validators.required]
  });
}

submit(): void {
  if (this.processing || this.form?.invalid) return;
  this.processing = true;

  const userForm: UserForm = this.form?.getRawValue();
  this.userAuthService
    .userLoginUsingPOST(userForm)
    .subscribe({
      next: res => {
        this.localStorageService.store('user', res);
        this.router.navigateByUrl(UrlHelper.expense());
      },
      error: (err: HttpErrorResponse) => {
        alertError(this.messageService, err.error);
        this.processing = false;
      },
      complete: () => this.processing = false
    });
}

navigateToSignUpPage(): void {
  this.router.navigateByUrl(
    UrlHelper.signUp()
  );
}
```

```
/**
 *
 * @param {{ username: String, password: String }} form
 * @returns { UserAuthDTO } UserAuthDTO
 * @throws ErrorDTO
 */
async function authenticateUser(form) {
    const user = await User.findOne({ username: form.username });

    if (!user) {
        throw new ErrorDTOBuilder()
            .setStatus(HTTP_STATUS.NOT_FOUND)
            .setMessage('User not found.')
            .build();
    }

    if (await bcrypt
        .compare(form.password, user.password) === false) {
        throw new ErrorDTOBuilder()
            .setStatus(HTTP_STATUS.UNAUTHORIZED)
            .setMessage('Invalid Password')
            .build();
    }

    const token = generateJwtToken(user);

    return userAuthDTOFromUserAndToken(user, token);
}
```

Figure 33. Log in

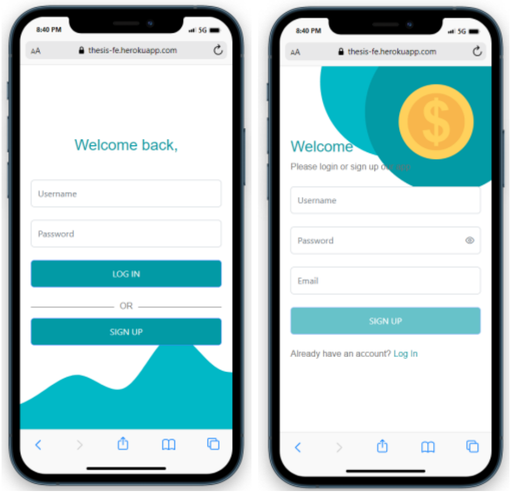


Figure 34. Sign Up/ Log in UI

After logging into the application successfully, the user is navigated to the home page. On the home page, the user can see the balance (Total Balance = Income – Expense) and create a new transaction via “+”-button on top right of the screen, as shown in Figure 35.

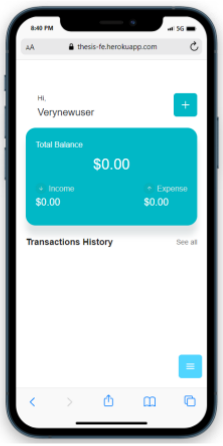


Figure 35. Home page interface with Quick Add function

The user can also navigate through the menu-button to other pages. The menu-button is located on the bottom right, in Figure 36. For the menu, the component <p-sidebar> was utilized.

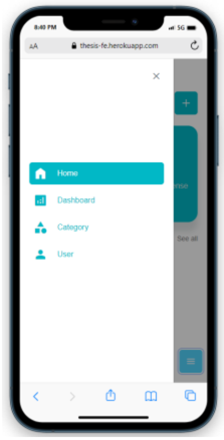
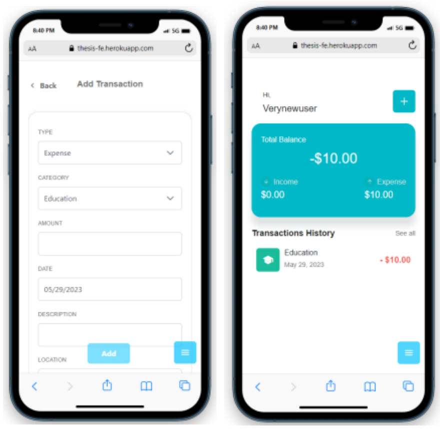


Figure 36. Menu side bar

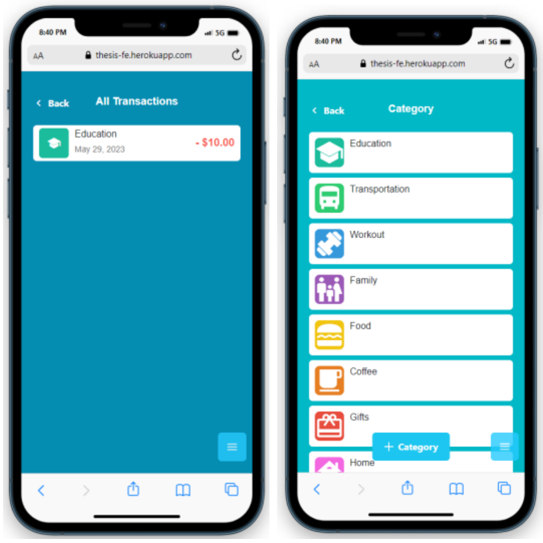
With quick add transaction, the application will navigate itself to the component expense-form.component.ts . Like with other forms, Validator was used to bind the conditions validity for the form (here is at least the AMOUNT must have a value). In the add transaction interface the user defines the transaction information, type, category, amount, date & description, and images, as shown in Figure 37.


8:40 PM
thesis-fe.herokuapp.com
Back Add Transaction
TYPE Expense
CATEGORY Education
AMOUNT
DATE 05/29/2023
DESCRIPTION
LOCATION
Add

Hi,
Verynewuser
Total Balance
-$10.00
Income $0.00
Expense $10.00
Transactions History See all
Education May 29, 2023 - $10.00

Figure 37. Add Transactions & Balance homepage

The user can click into an item to edit the transaction or use the button "See All" to have a look at all transactions available. Besides, the user can also navigate to the Category page, shown in Figure 38.


8:40 PM
thesis-fe.herokuapp.com
Back All Transactions
Education May 29, 2023 - $10.00

Back Category
Education
Transportation
Workout
Family
Food
Coffee
Gifts
Home
+ Category

Figure 38. See all Transactions/ Category

A few categories are created as default for the user when a new account is created. The user can also create new categories by using the "+Category" button. FormGroup was also used to manage inputs for NewCategory. One category needs to have: name, colour (for chart purpose), icon and description (optional). The application also allows the user to edit an existing category by clicking the category. The interface for editing a category is shown in Figure 39.

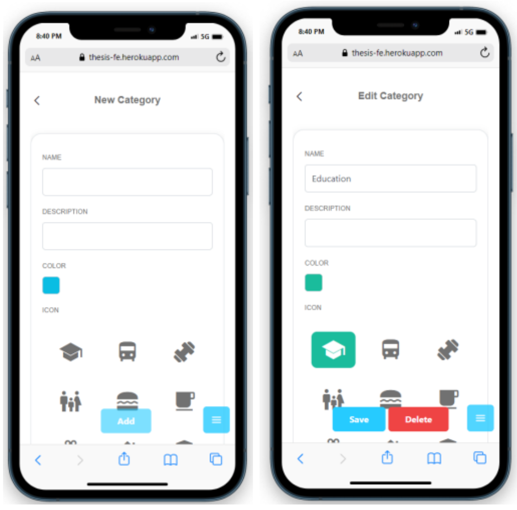


Figure 39. Create new Category/ Edit Category

In the Menu dashboard, the chart component from PrimeNG library is used as <p-chart>.FormGroup will also be used to manage the group button day/week/month/year, dropdown for filtering expense/income. After initialization of the form, valueChange event is set to listen these two fields, button and dropdown. Once the event is triggered, a request to get appropriate data is sent to the backend. The statistics pages are shown in Figure 40.

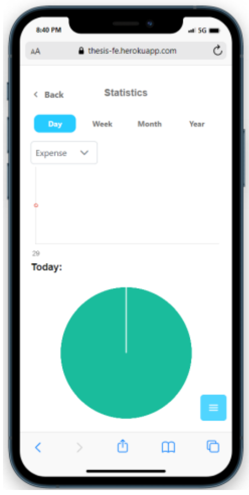


Figure 40. Statistics page with Charts

5.2 Heroku Deployment

Heroku is a cloud platform as a service (PaaS) that enables businesses to create, deliver, monitor, and scale applications. Heroku was created to alleviate infrastructure concerns and eliminate the need for learning about network administration, server configuration, or database tuning. Heroku removes the barriers so that developers can concentrate on creating excellent apps /19/

5.2.1 Deploy Backend Application

When deploying backend to Heroku, first a New MongoDB Cluster is created. The creation starts by clicking the “Build database” button. The type for the cluster needs to be selected. In this project, the Shared cluster was used. The cluster can be customized by selecting Share > AWS from menu and clicking the Create cluster -button. It is important at least to define the authentication for the cluster. In this project, the username and password were defined by clicking Create user -button. Connections also need to be specified. Once all settings are done, the window is closed by clicking Finish and Close button, shown in Figure 41

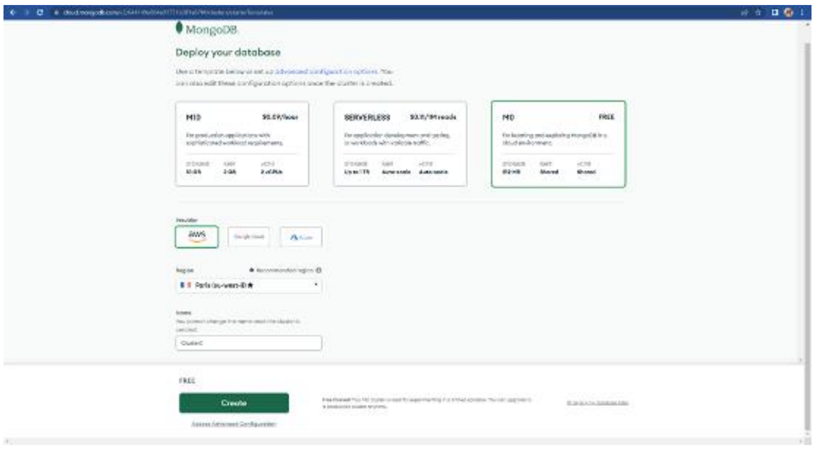


Figure 41. Customize Cluster

A connection string for the new MongoDB Cluster is needed to enable connection from the application. The connection string is stored into Heroku, it can be viewed by clicking the Connect-button, as shown in Figure 42. The connection string is stored under the Connect your application -tab. In the application settings, the

Connection string is defined inside the .env-file, the format of the string is shown in Figure 43.

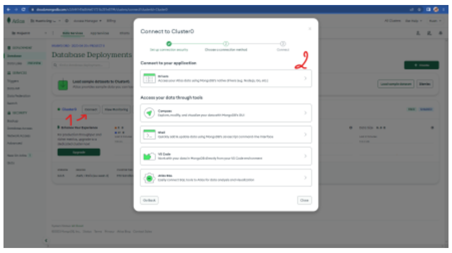
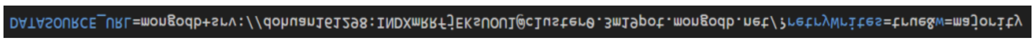


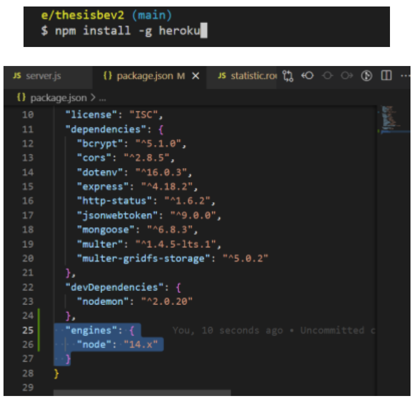
Figure 42. Get connection string



```
DATASOURCE_URL=mongodb+srv://[illegible]@cluster0.3wlebot.mongodb.net/?retryWrites=true&w=majority
```

Figure 43. Connection string is stored in .env file

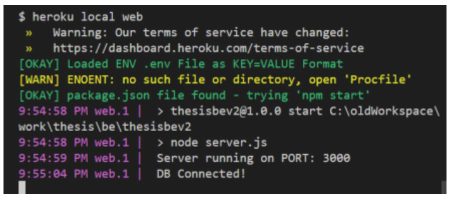
The Heroku Command Line Interface (CLI) enables the creation and management of Heroku applications directly from the terminal. It is an essential part of using Heroku. Heroku CLI is installed with the command npm install -g heroku. Json-package defines which scripts are used to start special to startscript. Finally, the application is started on port 3000 (Figure 45)



```
e/thesisbev2 (main)
$ npm install -g heroku
```

```
    "license": "ISC",
    "dependencies": {
      "bcrypt": "^5.1.0",
      "cors": "^2.8.5",
      "dotenv": "^16.0.3",
      "express": "^4.18.2",
      "http-status": "^1.6.2",
      "jsonwebtoken": "^9.0.0",
      "mongoose": "^6.8.3",
      "multer": "^1.4.5-lts.1",
      "multer-gridfs-storage": "^5.0.2"
    },
    "devDependencies": {
      "nodemon": "^2.0.20"
    },
    "engines": {          You, 10 seconds ago • Uncommitted c
      "node": "14.x"
    }
  }
```

Figure 44. Specify Node.js version used



```
$ heroku local web
 »   Warning: Our terms of service have changed:
 »   https://dashboard.heroku.com/terms-of-service
[OKAY] Loaded ENV .env File as KEY=VALUE Format
[WARN] ENOENT: no such file or directory, open 'Procfile'
[OKAY] package.json file found - trying 'npm start'
9:54:58 PM web.1 |  > thesisbev2@1.0.0 start C:\oldWorkspace\
work\thesis\be\thesisbev2
9:54:58 PM web.1 |  > node server.js
9:54:59 PM web.1 |  Server running on PORT: 3000
9:55:04 PM web.1 |  DB Connected!
```

Figure 45. Run Node.js and MongoDB app locally

The creation of a new application to Heroku is relatively easy, as shown in Figure 46. The creation starts by clicking the button “Create new app”. In creation, information, such as name and port for the application, is defined, as shown in Figure 47. A Git repository using a new or existing git-directory needs to be defined to enable continuous integration. In Heroku, configuration can be seen from ‘Setting’ - ‘Reveal Config Vars’ -view. The host IP address needs to be added to the Network Permission List. It gives the host access to the created cluster. Once all the settings are done, the application is deployed to Heroku, as shown in Figure 48.

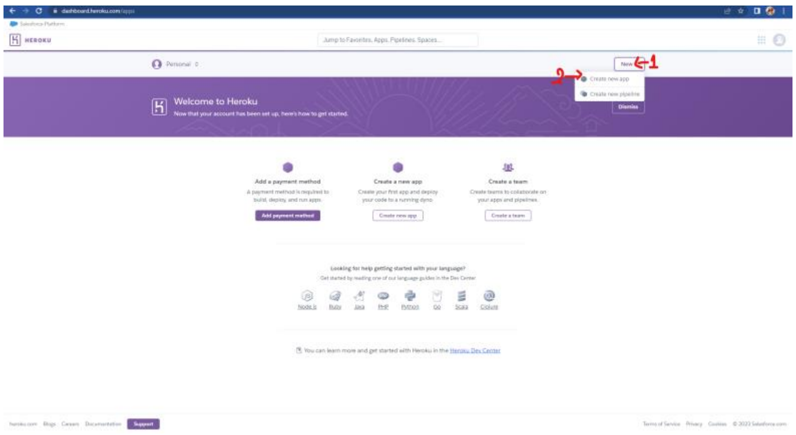
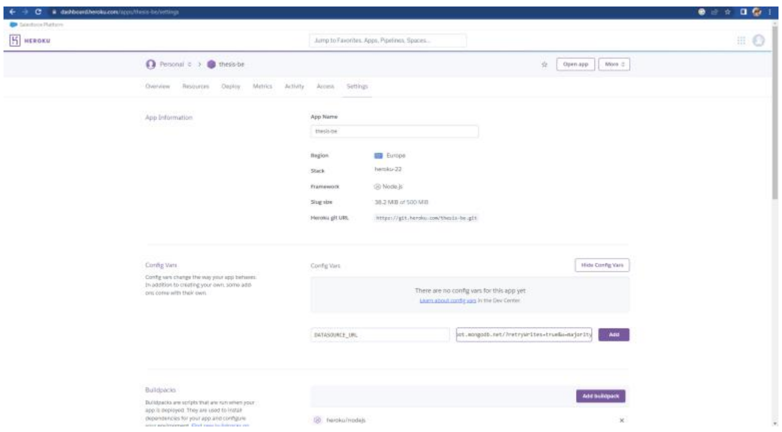

dashboard.heroku.com/apps
HEROKU
Jump to Favorites, Apps, Pipelines, Spaces...
Personal
New
1
2
Create new app
Create new pipeline
Welcome to Heroku
Now that your account has been set up, here's how to get started.
Dismiss
Add a payment method
A payment method is required to build, deploy, and run apps.
Add payment method
Create a new app
Create your first app and deploy your code to a running dyno.
Create new app
Create a team
Create teams to collaborate on your apps and pipelines.
Create a team
Looking for help getting started with your language?
Get started by reading one of our language guides in the Dev Center
Node.js
Ruby
Java
PHP
Python
Go
Scala
Clojure
You can learn more and get started with Heroku in the Heroku Dev Center
Support

Figure 46. Create new application



App Information

App Name: thesis-be

Region: Europe
Stack: heroku-22
Framework: Node.js
Slug size: 38.2 MiB of 500 MiB
Heroku git URL: https://git.heroku.com/thesis-be.git

Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

There are no config vars for this app yet
Learn about config vars in the Dev Center

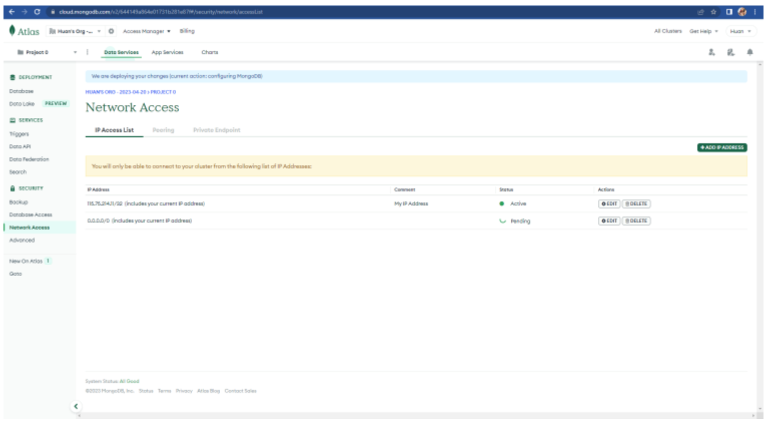
DATASOURCE_URL | ot.mongodb.net/?retryWrites=true&w=majority

Buildpacks

Buildpacks are scripts that are run when your app is deployed. They are used to install dependencies for your app and configure

heroku/nodejs

Figure 47. Host IP address is added



We are deploying your changes (current action: configuring MongoDB)

HUAN'S ORG - 2023-04-20 › PROJECT 0

Network Access

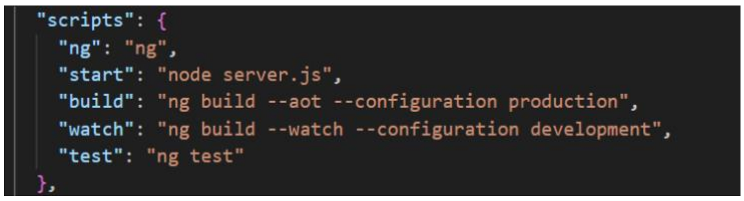
You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
115.75.214.1/32 (includes your current IP address)	My IP Address	● Active	EDIT DELETE
0.0.0.0/0 (includes your current IP address)		Pending	EDIT DELETE

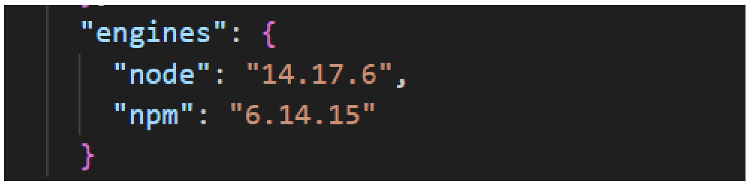
Figure 48. Application successfully deployed

5.2.2 Front-End Application

To setup the local environment, a file package.json needs to be created. That file specifies basic environment and server configuration, as shown in Figure 49. Server.js file, as shown in Figure 50, is the starter file for the front-end application. AS in software projects, all source files are stored in version control system. In this project, GitHub was used. By pushing files into Github, application is automatically configured to be built, as shown in Figure 51

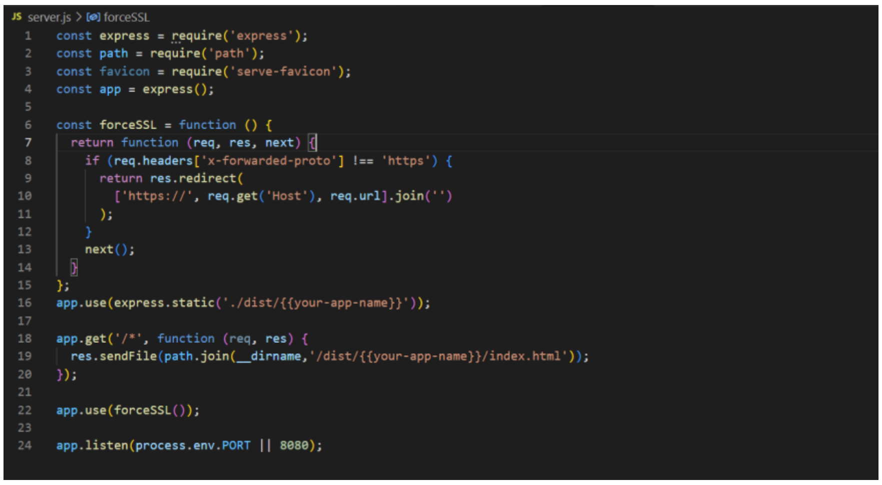


```
"scripts": {
  "ng": "ng",
  "start": "node server.js",
  "build": "ng build --aot --configuration production",
  "watch": "ng build --watch --configuration development",
  "test": "ng test"
},
```



```
"engines": {
  "node": "14.17.6",
  "npm": "6.14.15"
}
```

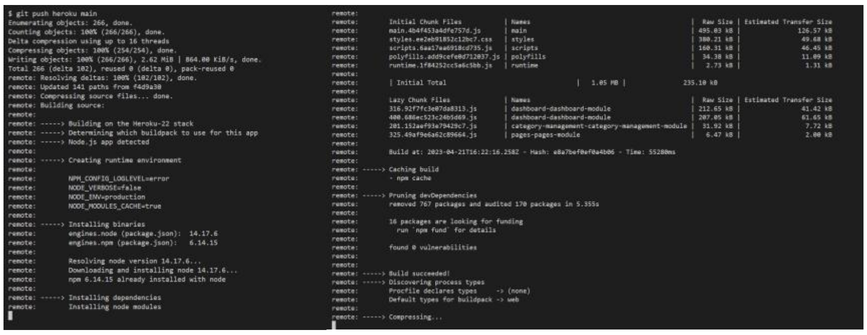
Figure 49. Setting up environment & configuration



server.js > forceSSL

```javascript
const express = require('express');
const path = require('path');
const favicon = require('serve-favicon');
const app = express();

const forceSSL = function () {
  return function (req, res, next) {
    if (req.headers['x-forwarded-proto'] !== 'https') {
      return res.redirect(
        ['https://', req.get('Host'), req.url].join('')
      );
    }
    next();
  }
};
app.use(express.static('./dist/{{your-app-name}}'));

app.get('/*', function (req, res) {
  res.sendFile(path.join(__dirname,'/dist/{{your-app-name}}/index.html'));
});

app.use(forceSSL());

app.listen(process.env.PORT || 8080);
```

Figure 50. Configuring file server.js

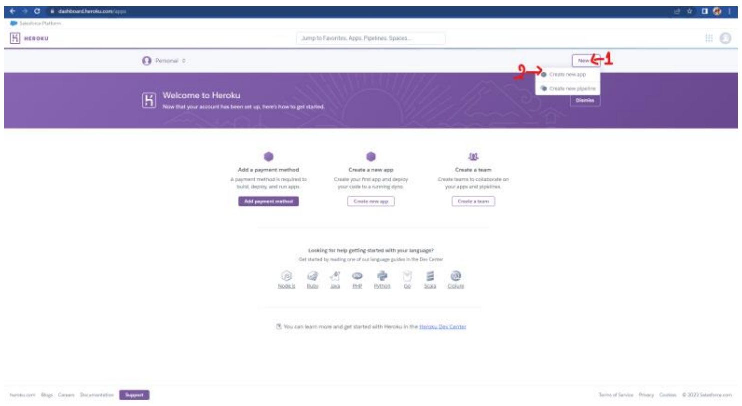


```
$ git push heroku main
Enumerating objects: 266, done.
Counting objects: 100% (266/266), done.
Delta compression using up to 16 threads
Compressing objects: 100% (254/254), done.
Writing objects: 100% (266/266), 2.62 MiB | 864.00 KiB/s, done.
Total 266 (delta 102), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (102/102), done.
remote: Updated 141 paths from f4d9a38
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-22 stack
remote: -----> Determining which buildpack to use for this app
remote: -----> Node.js app detected
remote:
remote: -----> Creating runtime environment
remote:
remote:        NPM_CONFIG_LOGLEVEL=error
remote:        NODE_VERBOSE=false
remote:        NODE_ENV=production
remote:        NODE_MODULES_CACHE=true
remote:
remote: -----> Installing binaries
remote:        engines.node (package.json):  14.17.6
remote:        engines.npm (package.json):   6.14.15
remote:
remote:        Resolving node version 14.17.6...
remote:        Downloading and installing node 14.17.6...
remote:        npm 6.14.15 already installed with node
remote:
remote: -----> Installing dependencies
remote:        Installing node modules
```

```
remote:
remote:        Initial Chunk Files           | Names                                         |  Raw Size | Estimated Transfer Size
remote:        main.4b4f453a4dfe757d.js      | main                                          | 495.03 kB |               126.57 kB
remote:        styles.ee2eb91852c12bc7.css   | styles                                        | 380.21 kB |                49.68 kB
remote:        scripts.6aa17ea6918cd735.js   | scripts                                       | 160.31 kB |                46.45 kB
remote:        polyfills.add9cefe0d712037.js | polyfills                                     |  34.38 kB |                11.09 kB
remote:        runtime.1f84252cc5a6c5bb.js   | runtime                                       |   2.73 kB |                 1.31 kB
remote:
remote:        | Initial Total                               |   1.05 MB |             235.10 kB
remote:
remote:        Lazy Chunk Files              | Names                                         |  Raw Size | Estimated Transfer Size
remote:        316.92f7fc3e07da8313.js       | dashboard-dashboard-module                    | 212.65 kB |                41.42 kB
remote:        400.686ec523c24b5d69.js       | dashboard-dashboard-module                    | 207.05 kB |                61.65 kB
remote:        201.152aef93e79429c7.js       | category-management-category-management-module |  31.92 kB |                 7.72 kB
remote:        325.49af9e6a62c89664.js       | pages-pages-module                            |   6.47 kB |                 2.00 kB
remote:
remote:        Build at: 2023-04-21T16:22:16.258Z - Hash: e8a7bef0ef0a4b06 - Time: 55280ms
remote:
remote: -----> Caching build
remote:        - npm cache
remote:
remote: -----> Pruning devDependencies
remote:        removed 767 packages and audited 170 packages in 5.355s
remote:
remote:        16 packages are looking for funding
remote:          run `npm fund` for details
remote:
remote:        found 0 vulnerabilities
remote:
remote:
remote: -----> Build succeeded!
remote: -----> Discovering process types
remote:        Procfile declares types     -> (none)
remote:        Default types for buildpack -> web
remote:
remote: -----> Compressing...
```

Figure 51. Running & pushing the project into GitHub

When deploying the application to Heroku, create new app is clicked, as shown in Figure 52. In the Deployment method, GitHub is chosen. After connecting GitHub to Heroku, the project created in GitHub is found and “Enable automatic deploy” can be chosen. After a successful deployment, the application can be accessed via URL https://thesis-fe.herokuapp.com as shown in Figure 53



Now that your account has been set up, here's how to get started.

A payment method is required to build, deploy, and run apps.

Create your first app and deploy your code to a running dyno.

Create teams to collaborate on your apps and pipelines.

Looking for help getting started with your language?

Get started by reading one of our language guides in the Dev Center

You can learn more and get started with Heroku in the Heroku Dev Center

Figure 52. Create new Front-end application

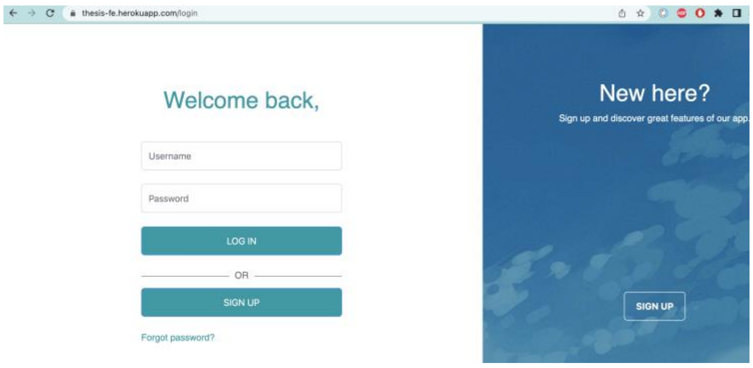


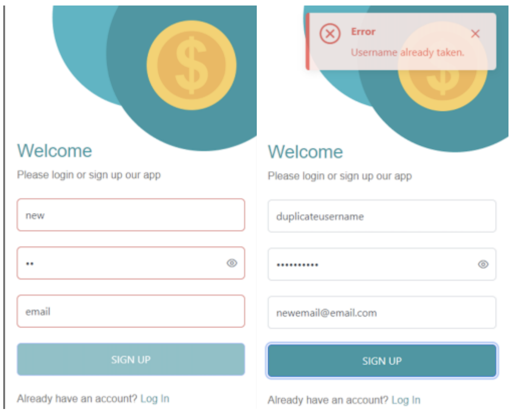
Figure 53. Application is deployed

5.3 Testing

This chapter describes the testing of all the possible error inputs, and shows in one of the contexts, that the application runs smoothly on a mobile device. Crucial functions are included: sign up, log in account, and some other features inside the application: amount adding, category editing.

5.3.1 Sign Up

This section tests: Register new account function with these possible scenarios. If any of input from the user is invalid or if the input is valid, but the username already exists. The expected result should be like this: the “Sign up” button will be disabled. The application pops up an error message: “Username is already taken!”, see Figure 54

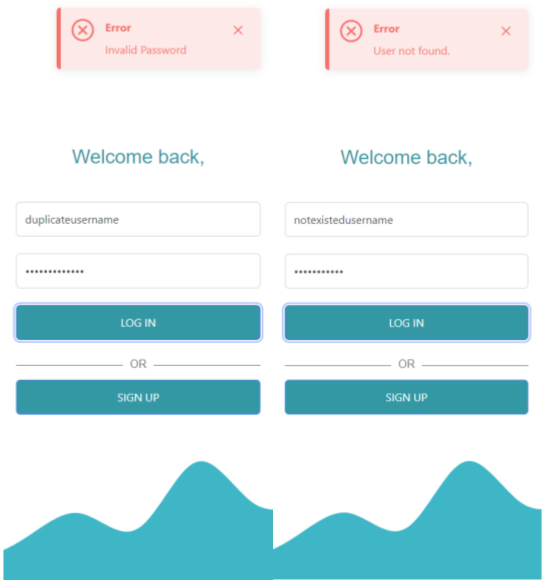

Error
Username already taken.

Welcome
Please login or sign up our app
new
email
SIGN UP
Already have an account? Log In
Welcome
Please login or sign up our app
duplicateusername
newemail@email.com
SIGN UP
Already have an account? Log In

Figure 54. Sign up page test

5.3.2 Log in Test

This section test: correct username, but wrong password/User does not exist. The expected result: the software pops an error message: “Invalid password”, “User not found” respectively for these scenarios, as seen in Figure 55 below


Error
Invalid Password

Error
User not found.

Figure 55. Log in page test

5.3.3 Expense with No Amount

This section tests: the add transaction function of the application. When the amount of the transaction is left “empty”, the expected result is the “Add” button is disabled, as shown in Figure 56 below.

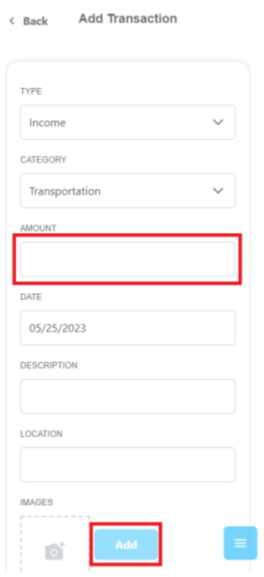


Figure 56. Expense Adding function with no amount value

5.3.4 Category without an icon/name

This section tests the Add category function. If the text “NAME” is left empty, or no icon had been chosen, the expected result is the “Add” button is disabled, as seen in Figure 57.

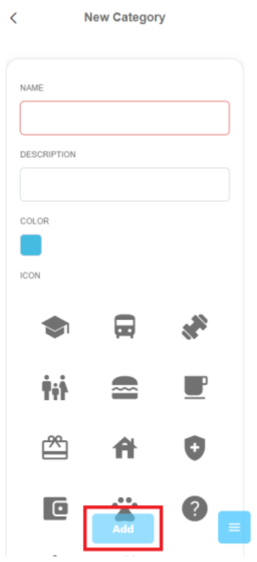


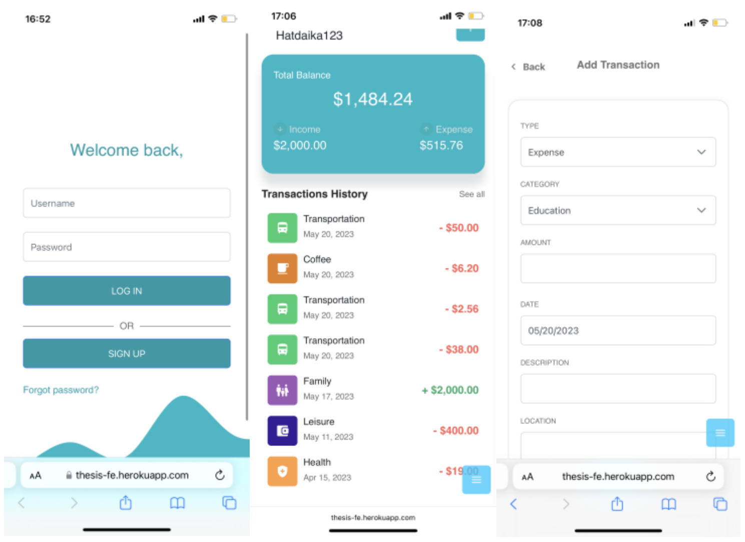
Figure 57. Create new Add/Edit

5.3.5 Some Other Noticeable Features of the Application

The application has also been tested with different browsers (Firefox, Google Chrome, Safari, Internet Explorer) and it worked out finely. It had been tested on a mobile device, specialized in an iPhone XS Max, and it runs perfectly.

Some figures of the mobile version were taken. In Figure 58, the Login page is visible for the user to log in, here one account was created already for testing purposes, namely “Hatdaika123”. As shown in the picture, after successfully logging in the application, a dashboard is displayed, with the “Total balance” of the user.

The user can see the income as well as the expense right below the balance. There is also a Transaction History where the user can see all incomes and expenses. These transactions are sorted into different categories: Family, transportation, coffee, … On the top right of the page, there a “+” sign where users can click to add transaction page. On the transaction page, users can add an amount, and categorize these spendings (Figure 58)


16:52
Welcome back,
Username
Password
LOG IN
OR
SIGN UP
Forgot password?
thesis-fe.herokuapp.com

17:06
Hatdaika123
Total Balance
$1,484.24
Income $2,000.00
Expense $515.76
Transactions History
See all
Transportation May 20, 2023 - $50.00
Coffee May 20, 2023 - $6.20
Transportation May 20, 2023 - $2.56
Transportation May 20, 2023 - $38.00
Family May 17, 2023 + $2,000.00
Leisure May 11, 2023 - $400.00
Health Apr 15, 2023 - $19
thesis-fe.herokuapp.com

17:08
Back
Add Transaction
TYPE
Expense
CATEGORY
Education
AMOUNT
DATE
05/20/2023
DESCRIPTION
LOCATION
thesis-fe.herokuapp.com

Figure 58. Log in page, Dashboard, Add transactions page demo

Figure 59 shows how the application looks in this interface: Category, Edit category. In this demo, the user can customize their category colour, icons (from Google Fonts>icons).

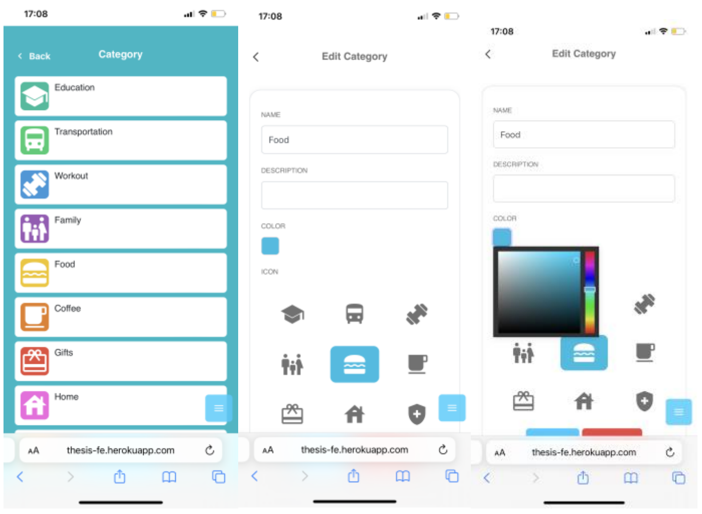


Figure 59. Category & Edit options demo

6 CONCLUSIONS

JavaScript is the most popular and commonly used programming language around the world. It adds versatility and speedy performance in any type of application. Being compatible with modern browsers and so much more makes JavaScript the frequent choice to developers. Angular is a powerful set of tools that support routing, guard, reusable components. Angular itself provides most of the tools so that it is ready to use. PrimeNG is a powerful component library that is easy to implement and use.

The thesis contributes to these experiences: route distribution using ExpressJS (BE application); organizing database: MongoDB is a no-SQL database, allowing flexible data saving; organizing files: each file takes their own role, i.e:expense.route only takes upcoming requests; expense. service will have functions for expense.

Lastly, the number of frameworks is developing all the time, for JavaScript. Whenever we have a framework that comes with useful additional functions, it becomes even more powerful.

REFERENCES

/1/ Wikipedia. JavaScript. Accessed May 20, 2023. https://en.wikipedia.org/wiki/JavaScript

/2/ Clark, Scott. 2010. “New Mobile Apps re Using HTML 5, CSS and JavaScript”. Accessed May 20, 2023. https://www.htmlgoodies.com/webmaster/html5-css-js-mobile-apps/

/3/ PrimeNG. “The Most Complete User Interface Suite for Angular”, “Why PrimeNG?”Accessed May 20, 2023. https://www.primefaces.org/primeng-v8-lts/#/

/4/ Wikipedia. 2023. “Bootstrap-CSS framework” Accessed May 25, 2023. https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework) .

/5/ Besant Technologies. “What is Express.JS?”, Accessed May 20, 2023. https://www.besanttechnologies.com/what-is-expressjs

/6/ @btroncone. 2020. “Learn RXJS, Introduction” Accessed May 20, 2023

https://www.learnrxjs.io/

/7/ “Google Fonts/Icons” official site. Accessed May 20,2023 https://fonts.google.com/icons?icon.query=add+a+photo

“Introducing Material Symbol”, fonts official site, accessed May 25, 2023 https://fonts.google.com/icons

/8/ “Lightbox2”, NPMJS official site, published Feb 2023. Accessed May 20, 2023 https://www.npmjs.com/package/lightbox2

/9/ “NGX-Webstorage” published 2022, Dec. Accessed May 21, 2023 https://www.npmjs.com/package/ngx-webstorage

/10/ “What is Bcrypt. How to use it to has passwords”, Shubham Sharma November 17, 2022. Accessed May 20, 2023 https://dev.to/documatic/what-is-bcrypt-how-to-use-it-to-hash-passwords-5c0g

/11/ “Introduction to JSON Web Tokens”, accessed May 20, 2023 https://jwt.io/introduction

/12/ ”dotenvTS” public, Published Oct 2022. Accessed May 20, 2023 https://www.npmjs.com/package/dotenv

/13/ “MomentJS”, published July 2022. Accessed May 20, 2023 https://www.npmjs.com/package/moment

/14/ “Introduction to Mongoose for MongoDB”, freecodecamp official site, published Feb 11, 2018. Accessed May 20, 2023 https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

/15/” Multer” published to publicity, May 2022. Accessed May 20, 2023 https://www.npmjs.com/package/multer?activeTab=readme

/16/ “Multer GridFS Storage” published May 2021. Accessed May 20, 2023 https://www.npmjs.com/package/multer-gridfs-storage

“GridFS”, MongoDB offical. Accessed May 26, 2023 https://www.mongodb.com/docs/manual/core/gridfs/

/17/ “Cross-origin resource” sharing last edited 10 February 2023. Accessed May 20, 2023, https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

/18/ “HTTP Status codes for Node”, published January 2023. Accessed May 20, 2023, https://www.npmjs.com/package/http-status

/19/ Ekekenta Odionyefe, “Deploy Node.js & MongoDB Application to Heroku”, Oct 12, 2022. Accessed May 17, 2023.https://blog.appsignal.com/2022/10/12/deploy-a-nodejs-and-mongodb-application-to-heroku.html

/20/ Angular. 2022. “The RXJS library”. Accessed May 25, 2023.https://angular.io/guide/rx-library#:~:text=RxJS%20(Reactive%20Extensions%20for%20JavaScript,asynchronous%20or%20callback%2Dbased%20code