|  |  |
| --- | --- |
| **Assignment** | |
| **Course Code** | CSC402A |
| **Course Name** | Web Architecture and Application Development |
| **Programme** | B.Tech |
| **Department** | CSE |
| **Faculty** | FET |

|  |  |
| --- | --- |
| **Name of the Student** | Satyajit Ghana |
| **Reg. No.** | 17ETCS002159 |
| **Semester/Year** | 07/2021 |
| **Course Leader(s)** | Mrs. Sahana P Shankar |



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Declaration Sheet | | | | | | | | |
| Student Name | Satyajit Ghana | | | | | | | |
| Reg. No | 17ETCS002159 | | | | | | | |
| Programme | B.Tech | | | | | Semester/Year | 07/2021 | |
| Course Code | CSC402A | | | | | | | |
| Course Title | Web Architecture and Application Development | | | | | | | |
| Course Date |  | | to |  | | | | |
| Course Leader | Mrs. Sahana P Shankar | | | | | | | |
| **Declaration**  The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly. | | | | | | | | |
| Signature of the Student | |  | | | | | Date |  |
| Submission date stamp  (by Examination & Assessment Section) | |  | | | | | | |
| Signature of the Course Leader and date | | | | | Signature of the Reviewer and date | | | |
|  | | | | |  | | | |

# Contents

[Declaration Sheet ii](#_Toc61915438)

[Contents iii](#_Toc61915439)

[1 Question 1 4](#_Toc61915440)

[1.1 Implementation of database with justification of relationships 4](#_Toc61915441)

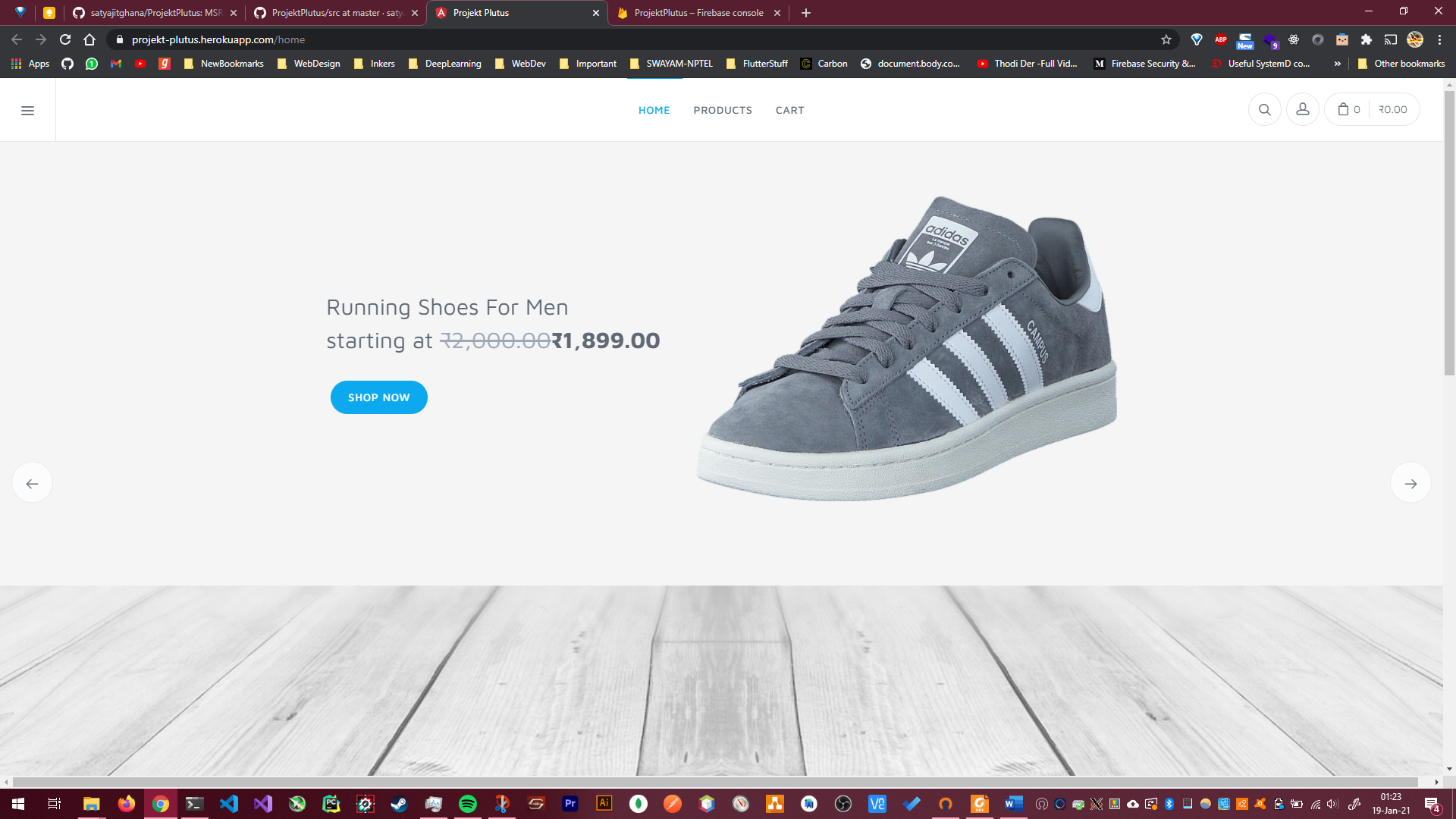
[1.2 Implementation of user registration 8](#_Toc61915442)

[1.3 Implementation of product management 12](#_Toc61915443)

[1.4 Implementation of sports accessories reservation for user 13](#_Toc61915444)

[Bibliography 22](#_Toc61915445)

# Question 1

Solution to Question No. 1

The deployed website can be viewed/tested at <https://projekt-plutus.herokuapp.com/>

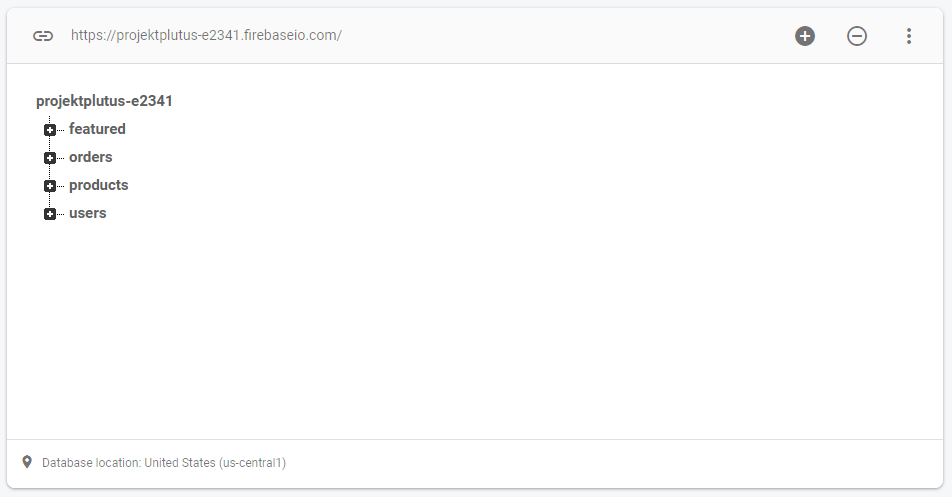
The source code can be viewed at https://github.com/satyajitghana/ProjektPlutus/

## Implementation of database with justification of relationships

For implementing the database Firebase Realtime Database was used, The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in real-time to every connected client.

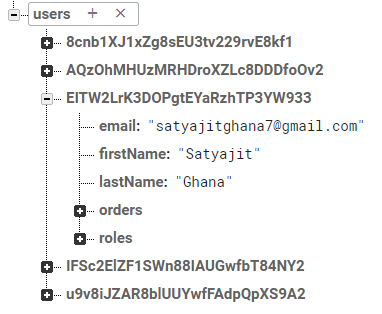
The Realtime Database is a NoSQL database and as such has different optimizations and functionality compared to a relational database. The Realtime Database API is designed to only allow operations that can be executed quickly.

The reason to choose a NoSQL database was to make the database more flexible, suppose we would like to add another field to the product later in the future, it won’t be possible in an SQL database due to its limitation, but we can do it in NoSQL!, below is the outline of the keys present in the database. Below are the root keys in our database.

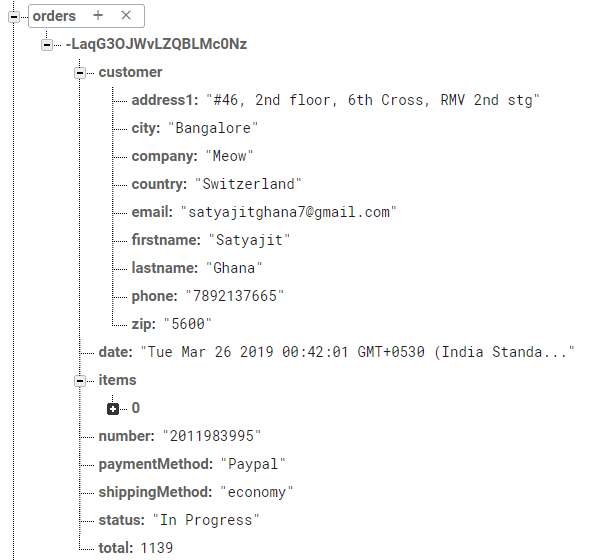


#### Users

These are both the admin and the normal users, they are identified using a unique id, and their access level is identified by the roles key

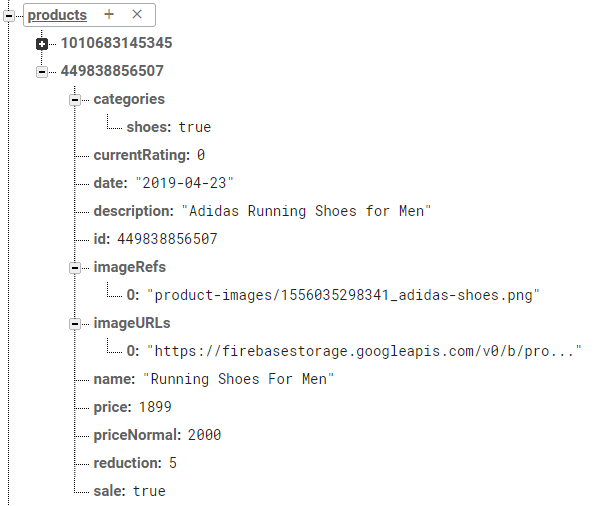


Each of the customer has a copy of their order they have made, we have already seen that there is a orders key in the root key, replication actually helps in Firebase Databases, although it takes up more space, it makes it easier to index items, let’s suppose we would like to retrieve all orders of a specific user, then instead of filtering the root orders key, we can simply query the specific user’ order in their key.



#### Products

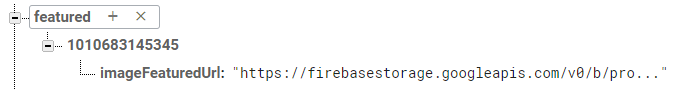
This key stores all the products available in our website, along with its rating, description, product image, price, discounted price, etc.



Each of the product is identified by its id, which is also the key of the product

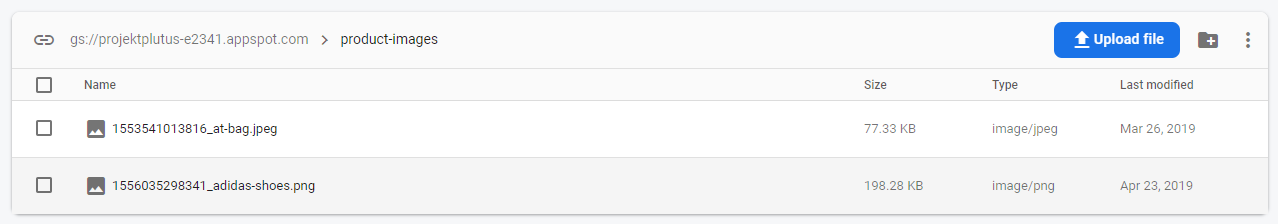
#### Featured

Featured are the products that show up in the landing page of the website



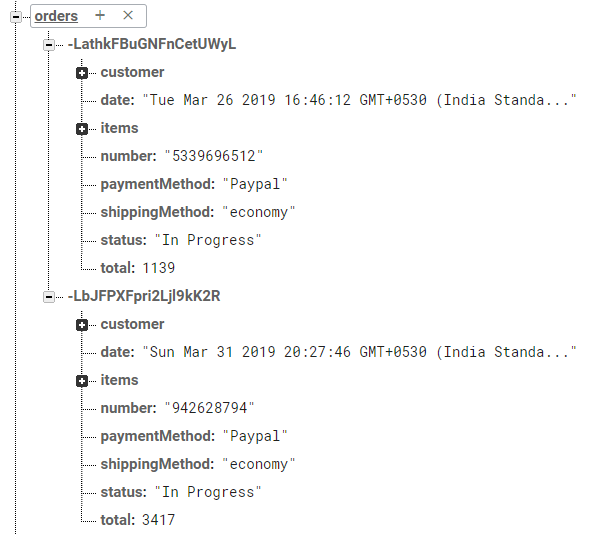
#### Storage

All of the products, have some images associated with them, these images are stores in Firebase Storage, the public URL from this is used in the database, so the client can query and fetch the image to be displayed for that product.



#### Orders

This key holds all the orders placed by all of the users, this can be used by the merchants to calculate their total sales, profits, etc. Each of the order is identified by its key, and the value contains the attributes of the order as discussed in Assignment 1.



## Implementation of user registration

The UI/HTML can be viewed at

<https://github.com/satyajitghana/ProjektPlutus/tree/master/src/app/account>

We’ll discuss some of the core logic involved here,

Since we are using angularfire2 we can leverage their API’s directly to authenticate users using Firebase Auth

auth.service.ts

*public* emailSignUp(email: string, password: string) {

*return* *this*.afAuth.auth

      .createUserWithEmailAndPassword(email, password)

      .then(

        (user) *=>* {

          console.log(user.user);

*this*.updateNewUser(user.user);

        },

        (error) *=>* {

*throw* error;

        }

      );

  }

  emailLogin(email: string, password: string) {

*return* *this*.afAuth.auth.signInWithEmailAndPassword(email, password).then(

      (user) *=>* {

*this*.updateNewUser(user);

      },

      (error) *=>* {

*throw* error;

      }

    );

  }

*public* signOut() {

*this*.afAuth.auth.signOut();

*this*.messageService.add('You have been logged out.');

  }

*public* updateProfile(userData: User) {

*this*.updateExistingUser(userData);

*this*.messageService.add('User profile has been updated!');

  }

*public* updatePassword(password: string) {

*return* *this*.afAuth.auth.currentUser

      .updatePassword(password)

      .then(() *=>* {

*this*.messageService.add('Password has been updated!');

      })

      .catch(*function*(error) {

*throw* error;

      });

  }

*public* updateEmail(email: string) {

*return* *this*.afAuth.auth.currentUser

      .updateEmail(email)

      .then(() *=>* {

*this*.updateExistingUser({ email: email });

*this*.messageService.add('User email have been updated!');

      })

      .catch(*function*(error) {

*throw* error;

      });

  }

*private* updateNewUser(authData) {

*const* userData = *new* User(authData);

*const* ref = *this*.db.object('users/' + authData.uid);

    ref

      .valueChanges()

      .pipe(

        take(1)

      )

      .subscribe((user) *=>* {

*if* (!user) {

          console.log(userData);

          ref.update(userData);

        }

      });

  }

user.model.ts

*export* *interface* Roles {

  admin: boolean;

}

*export* *class* User {

*public* email: string;

*public* photoURL?: string;

*public* roles?: Roles;

*public* firstName?: string;

*public* lastName?: string;

*public* password?: string;

*public* orders?: object;

*public* confirmPassword?: string;

*public* uid?: string;

*constructor*(authData) {

*this*.email = authData.email;

*this*.firstName = authData.firstName ? authData.firstName : '';

*this*.lastName = authData.lastName ? authData.lastName : '';

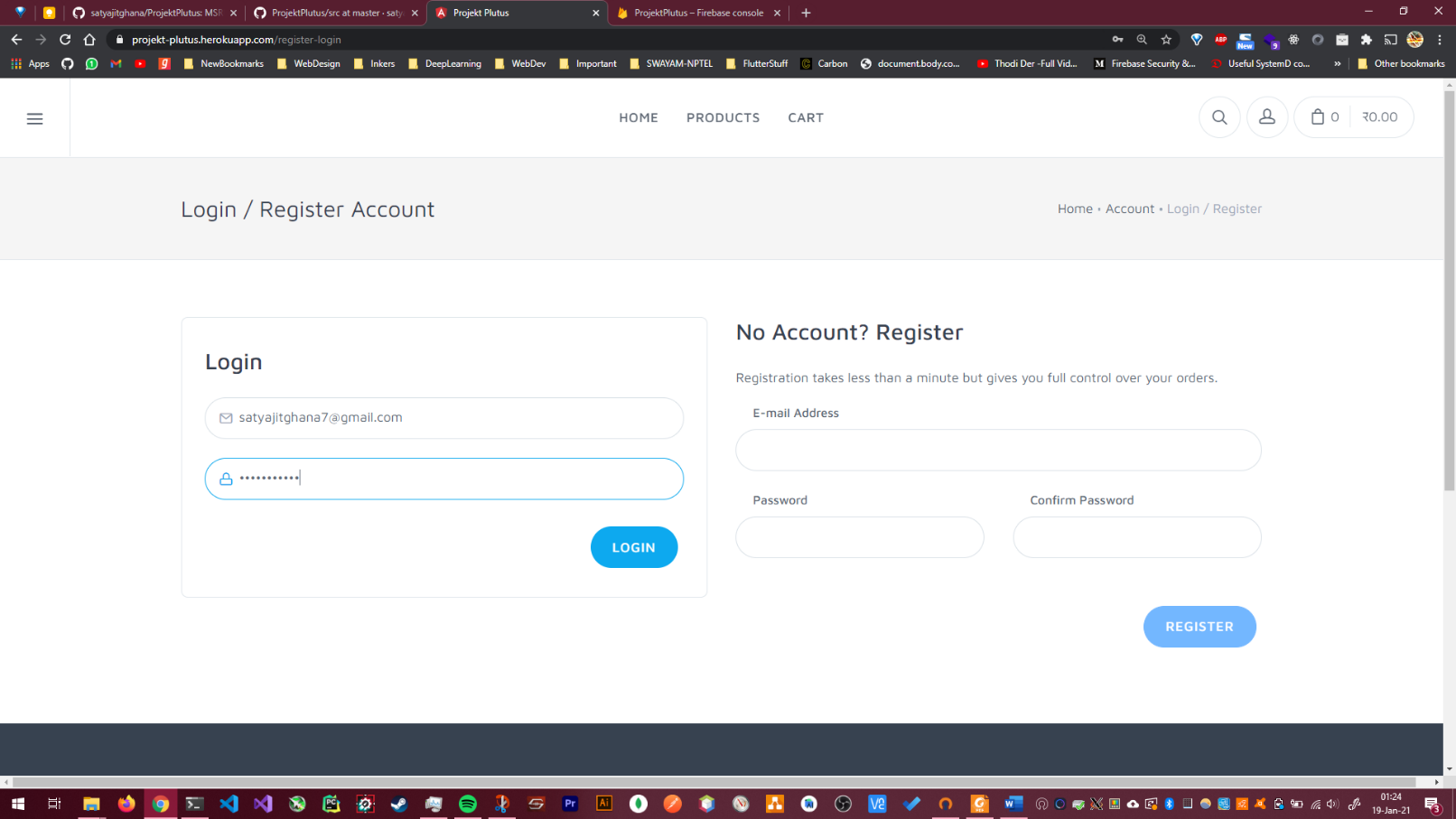
*this*.roles = {

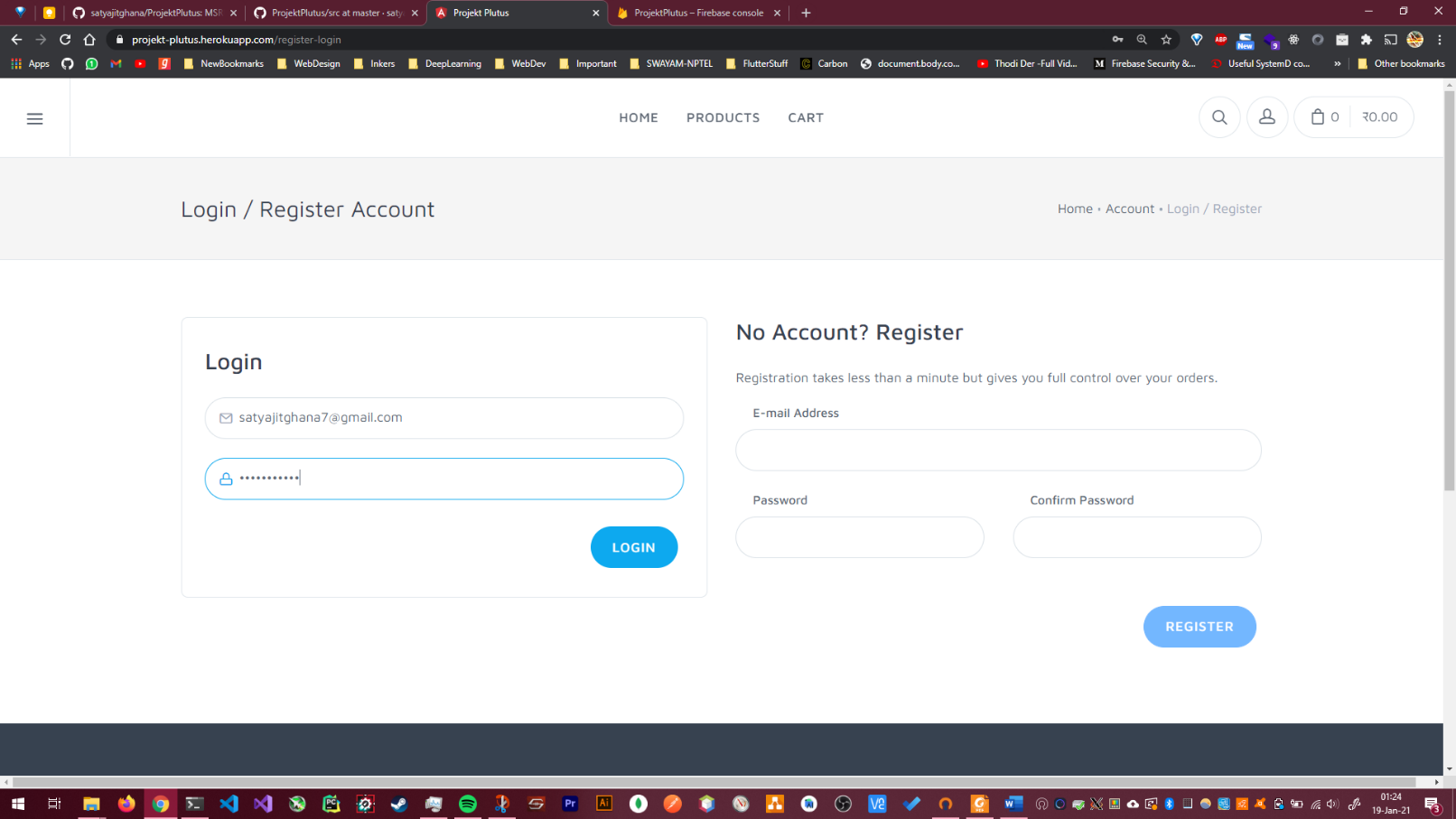
      admin: *false*

    };

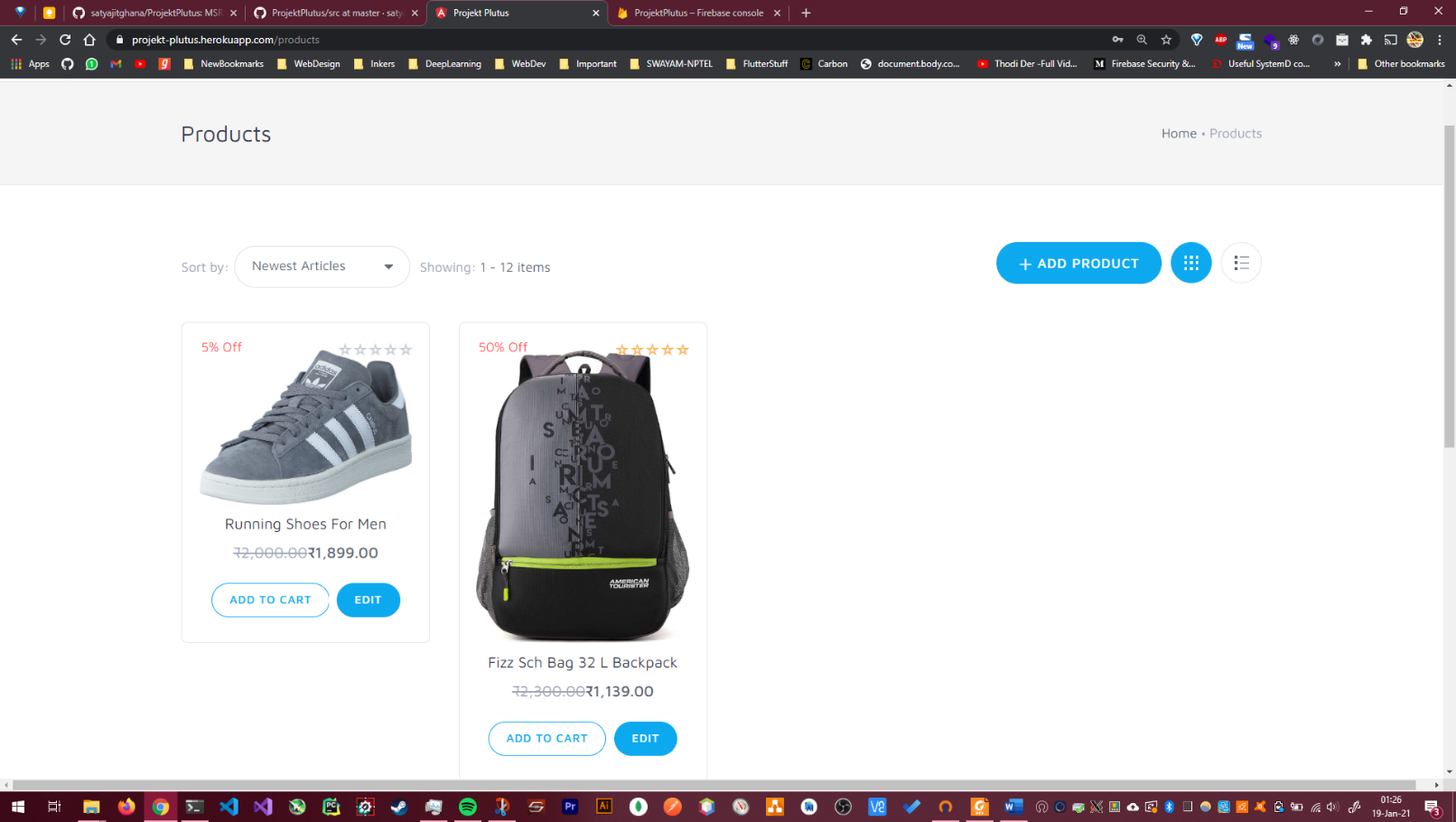
  }

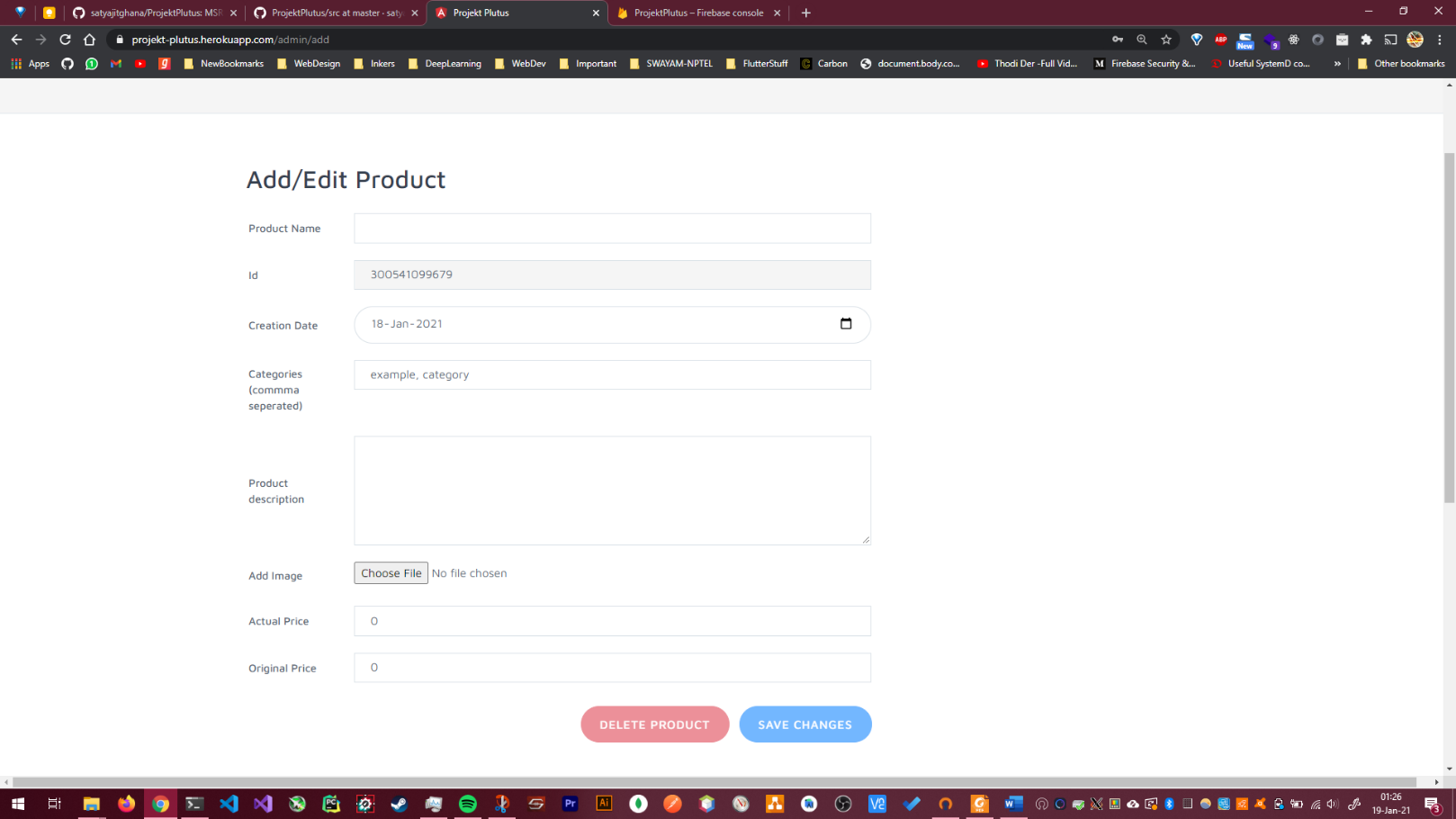
}

Here is the UI for User Login/Registration



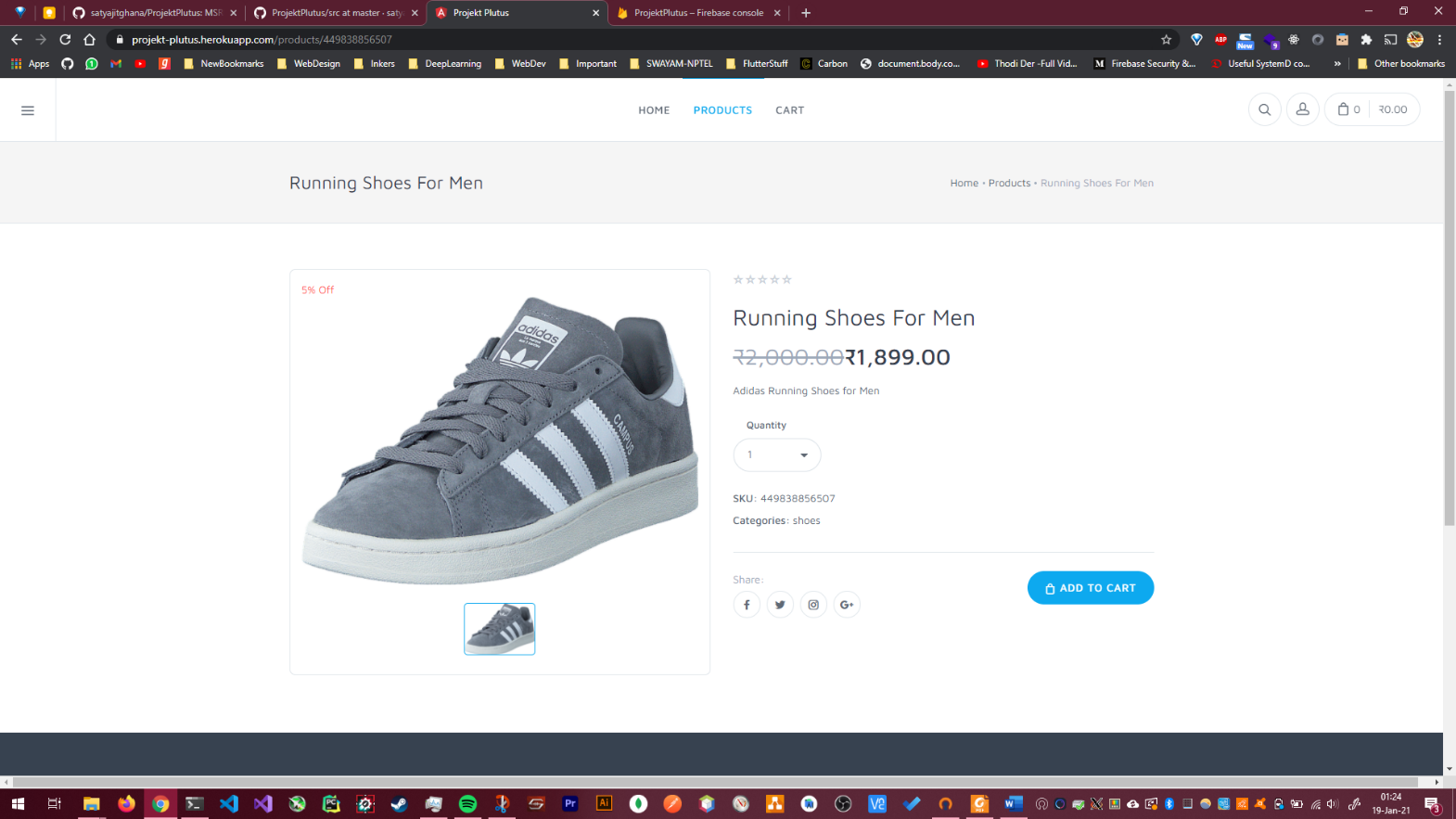
## Implementation of product management

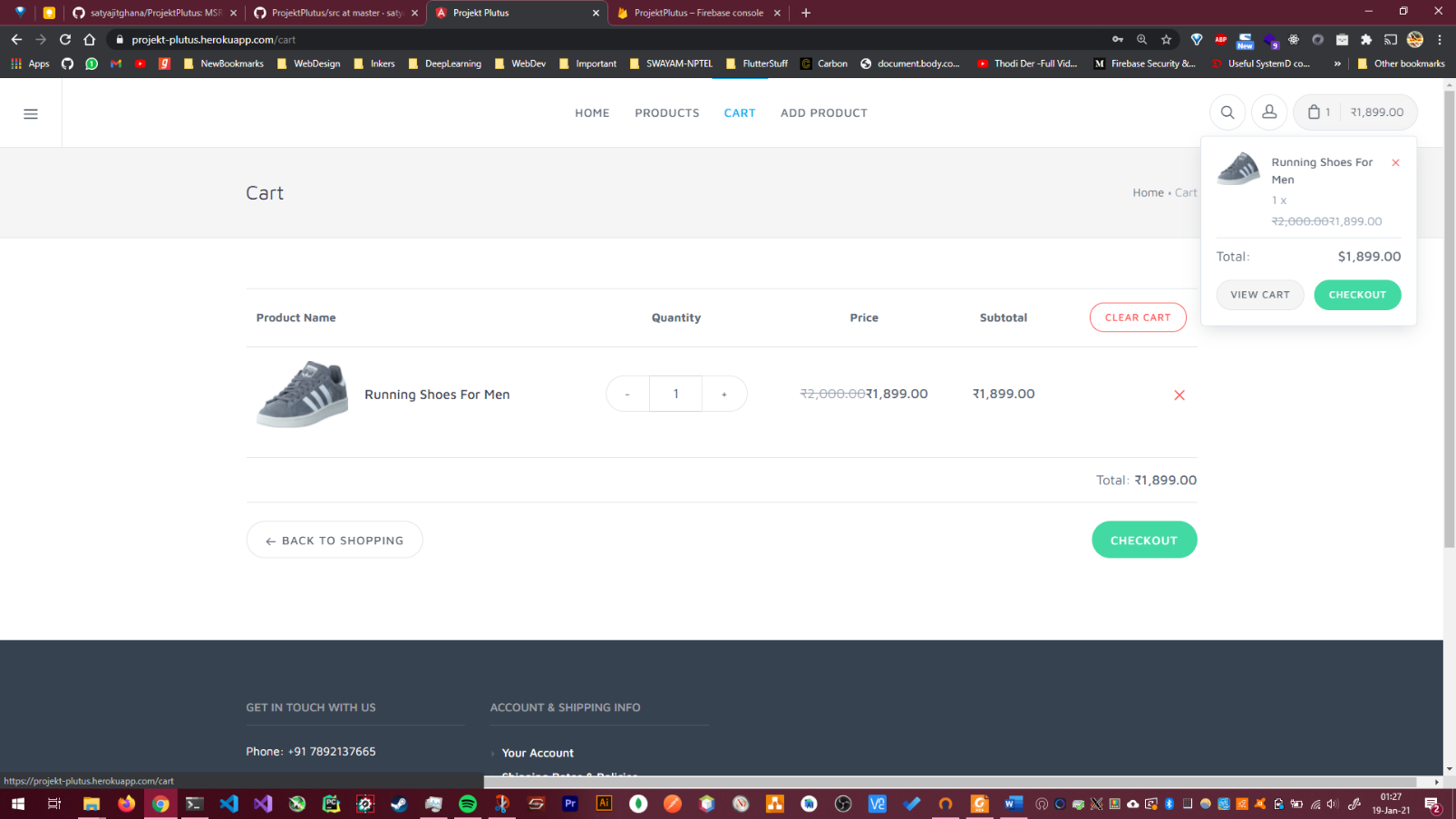
New Products can be added to the website by the merchant using this feature

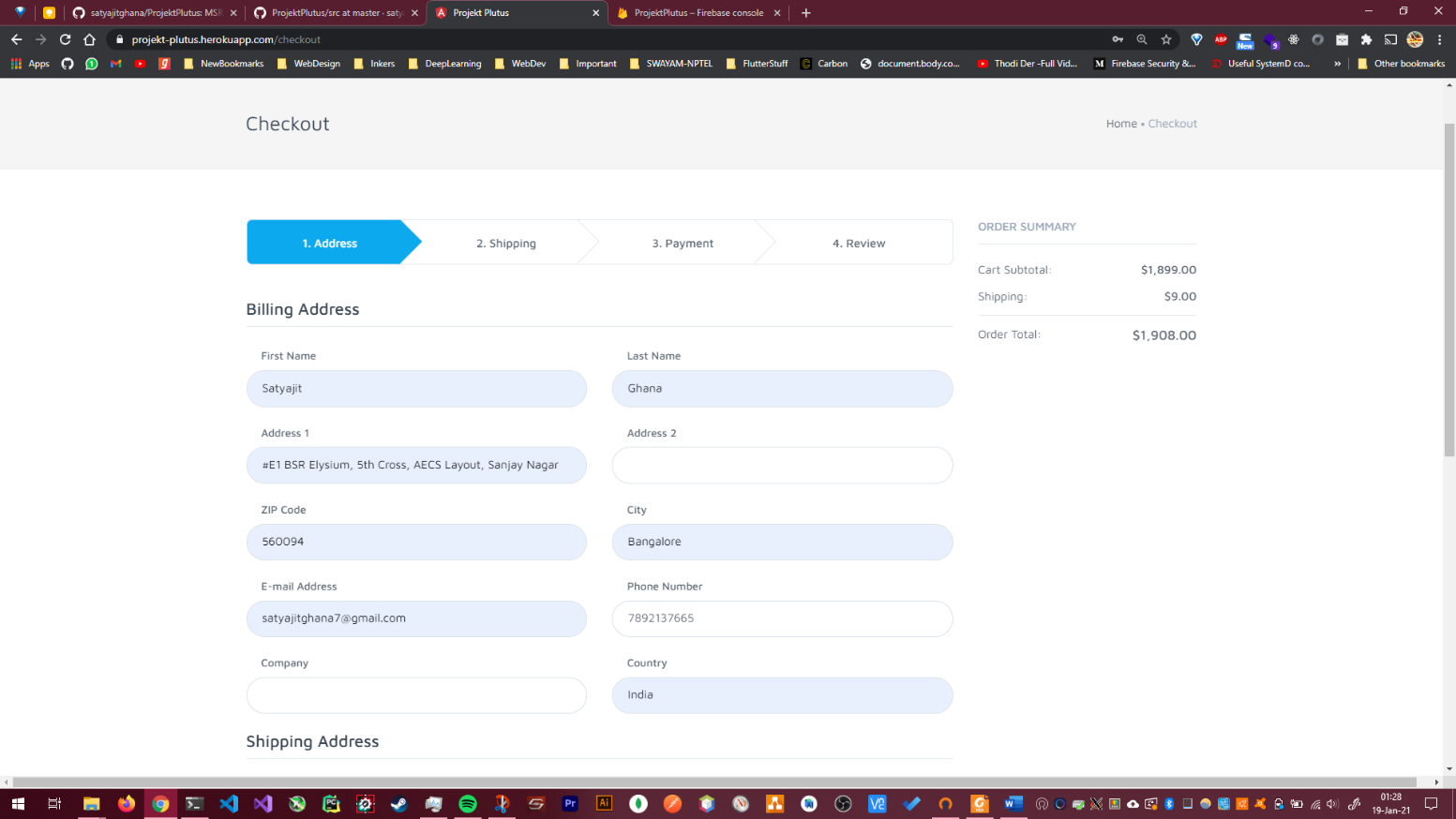


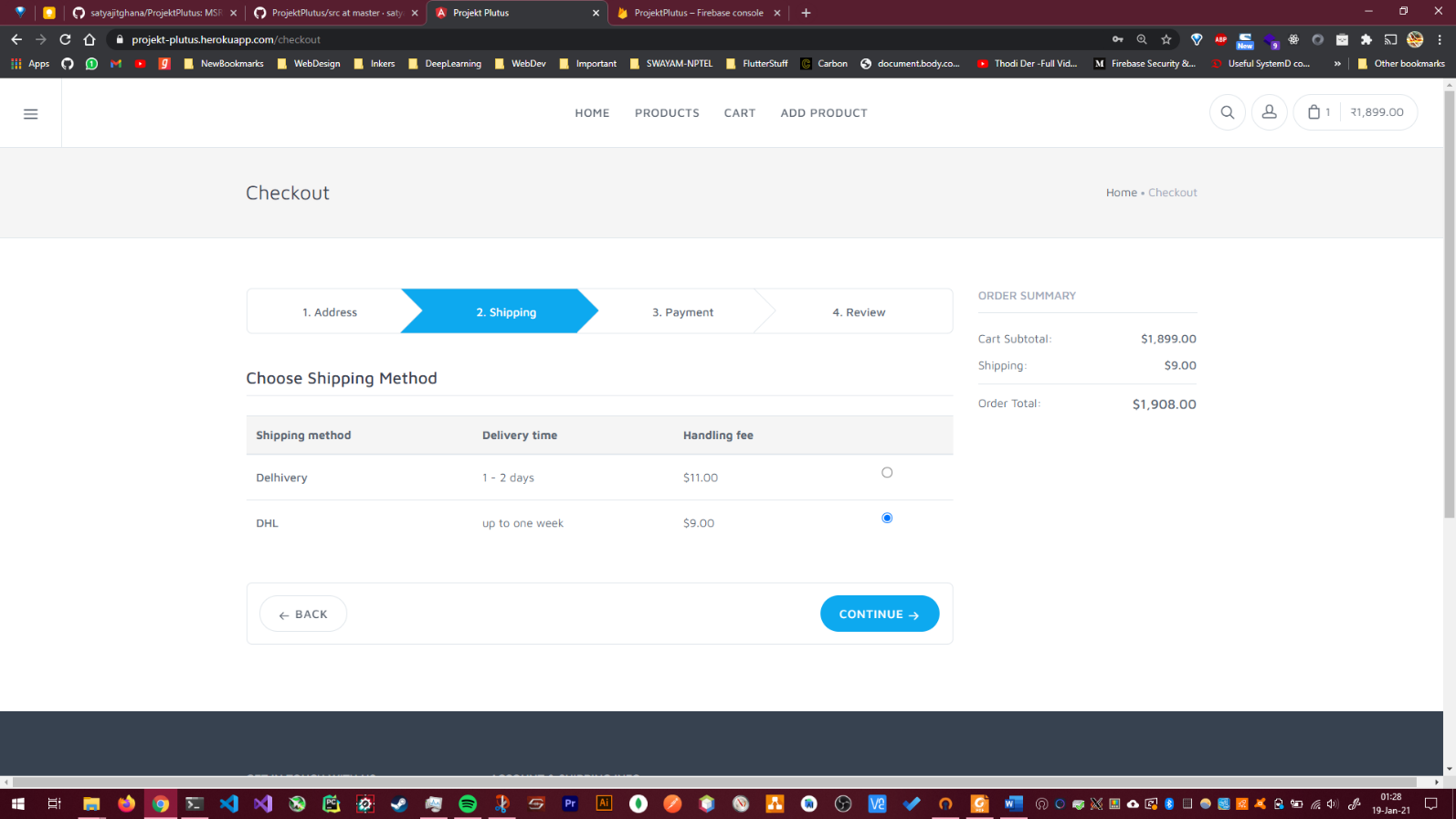
## Implementation of sports accessories reservation for user

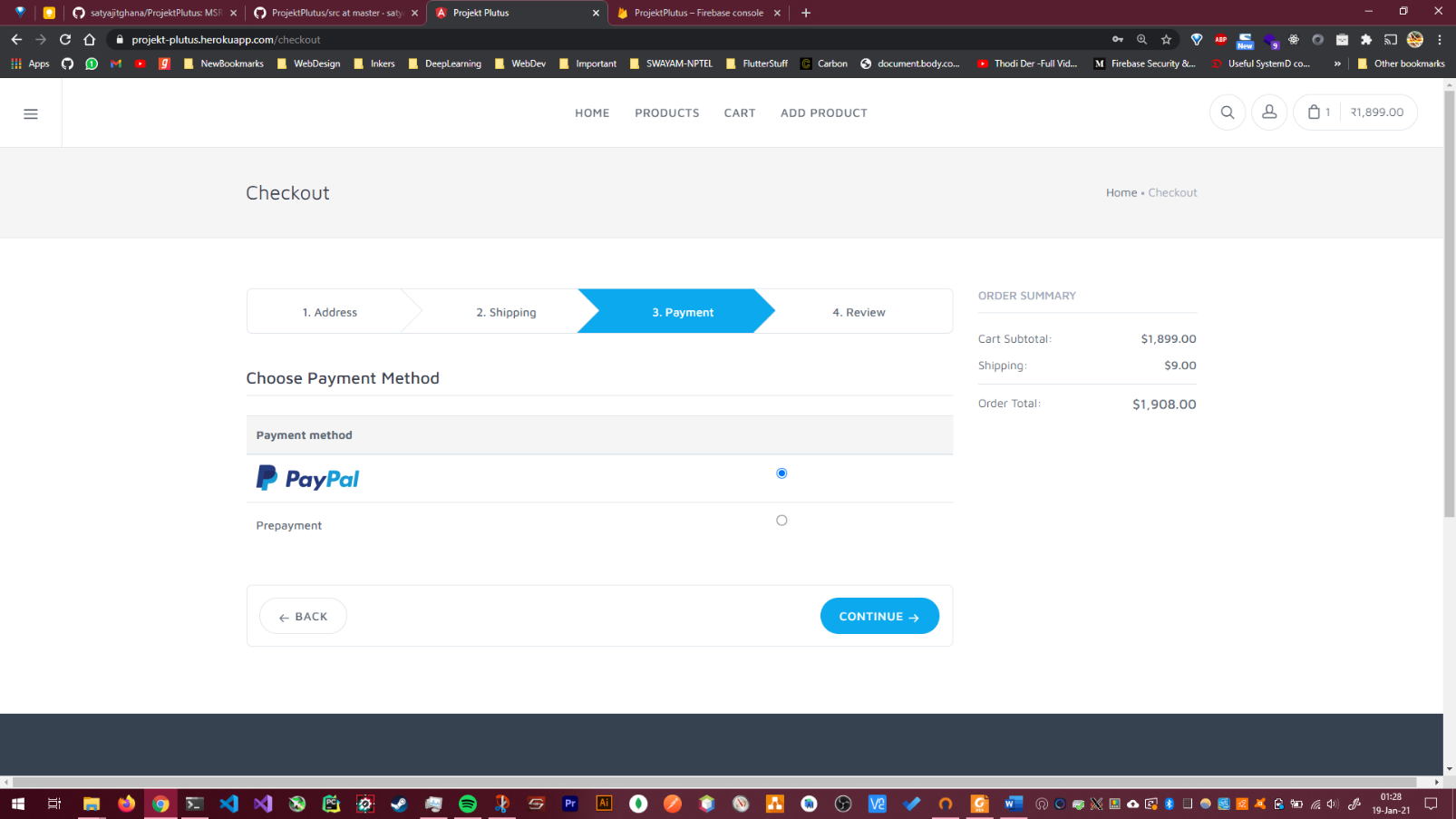
This shows the Product Purchase flow,

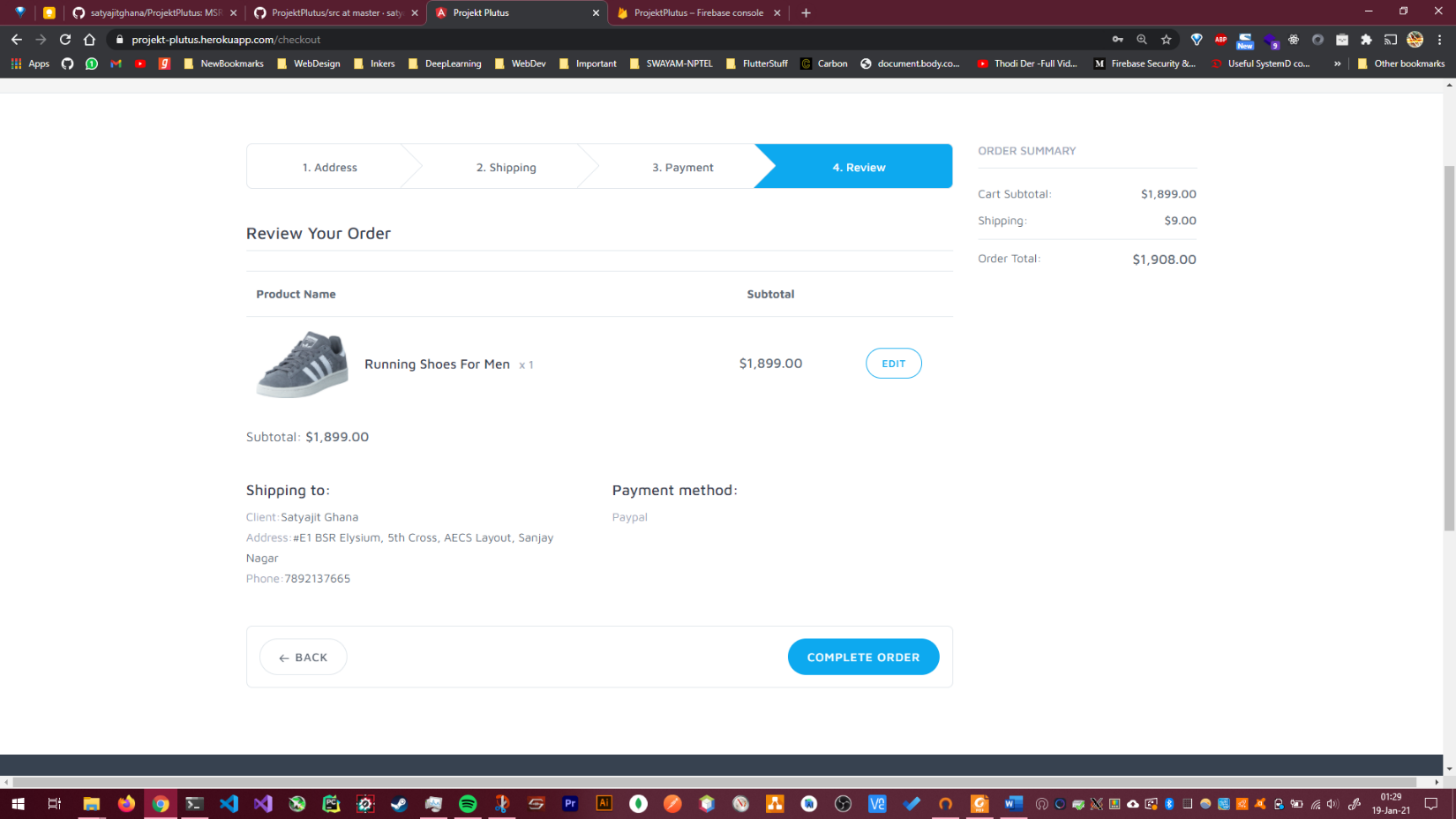
The user browser for a product, selects it, and then there’s an option to ADD TO CART

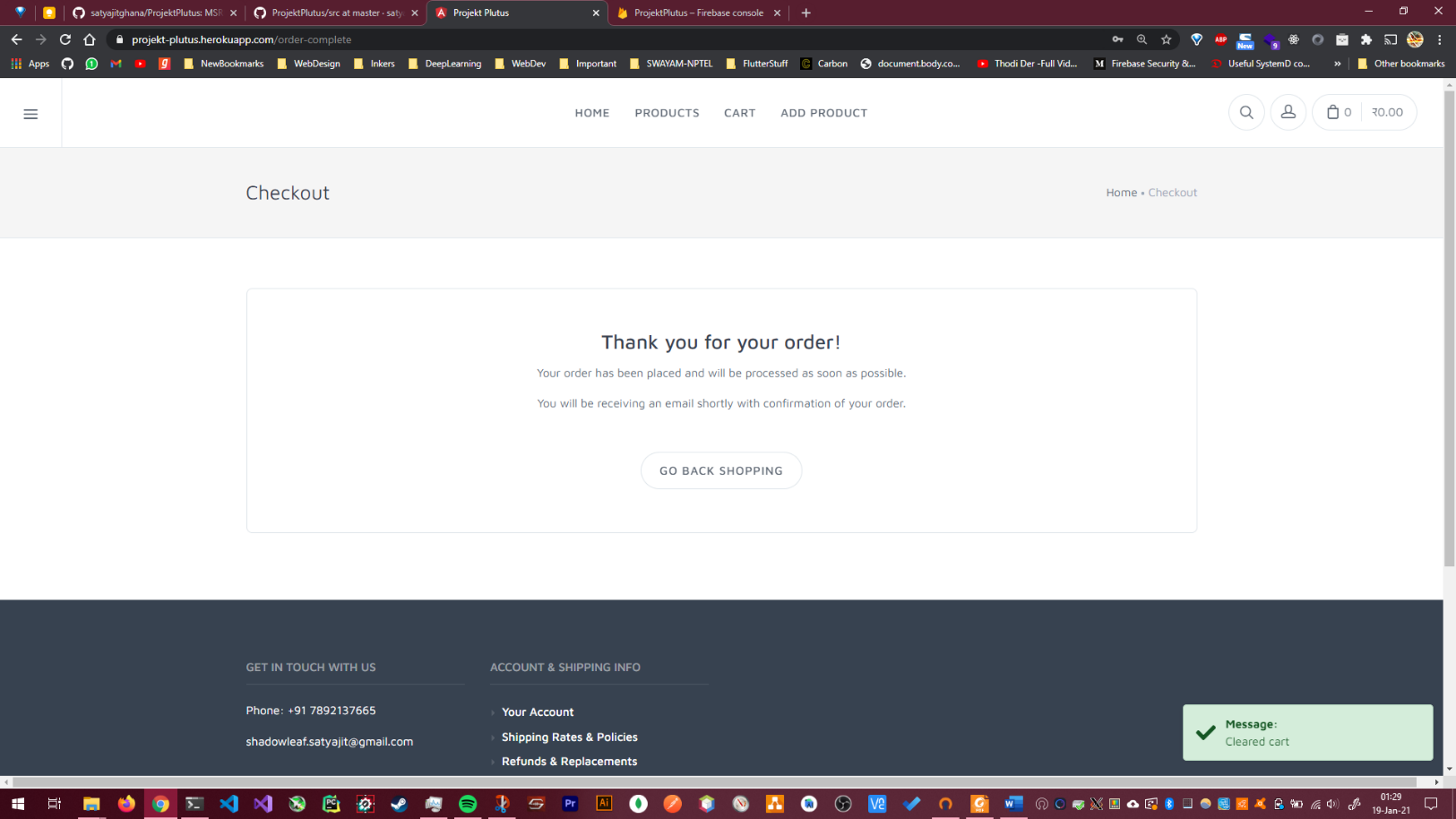
Clicking it adds the product to the cart at the top right

Now all the details for the Checkout are asked to the user, like Address, ZIP Code, City, Email, Phone Number, Company, Country

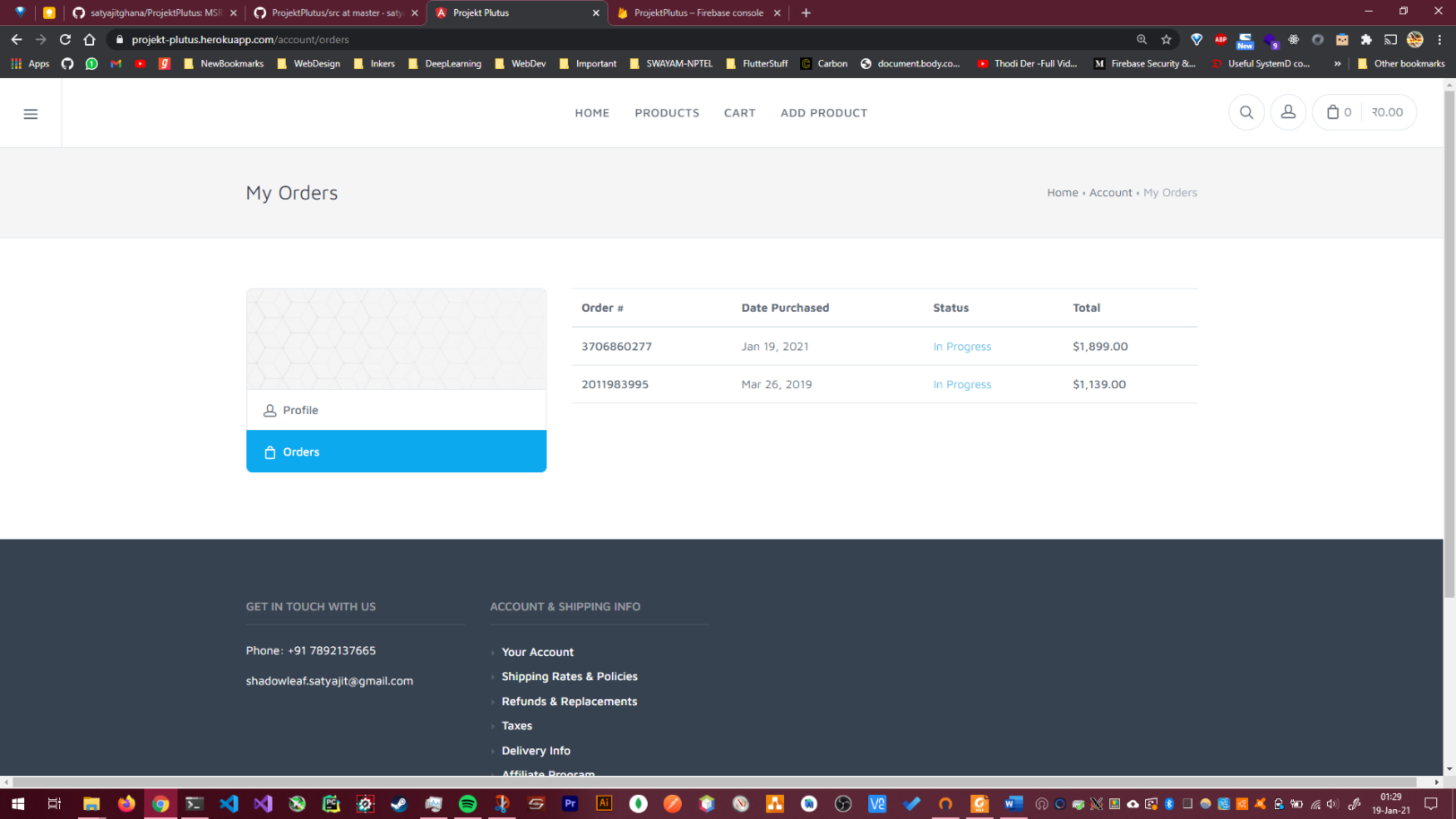
Now the Shipping method can be selected,

Once that is done, use user has to pay using PayPal

Once the payment method is selected, a final checkup is show, which shows the shipping address, product details, total price, and payment method.

Once payment is done, payment confirmation is shown

And the user can then view the order in their profile.



#### Implementation Specifics

The UI/HTML can be viewed at

<https://github.com/satyajitghana/ProjektPlutus/tree/master/src/app/checkout>

The cart is implemented by using an Injectable Service that keeps track of the products in the cart, and updates the total cart price as and when the products are added to the cart.

@Injectable()

*export* *class* CartService {

*// Init and generate some fixtures*

*private* cartItems: CartItem[];

*public* itemsChanged: EventEmitter<CartItem[]> = *new* EventEmitter<CartItem[]>();

*constructor*(*private* messageService: MessageService) {

*this*.cartItems = [];

  }

*public* getItems() {

*return* *this*.cartItems.slice();

  }

*// Get Product ids out of CartItem[] in a new array*

*private* getItemIds() {

*return* *this*.getItems().map(cartItem *=>* cartItem.product.id);

  }

*public* addItem(item: CartItem) {

*// If item is already in cart, add to the amount, otherwise push item into cart*

*if* (*this*.getItemIds().includes(item.product.id)) {

*this*.cartItems.forEach(*function* (cartItem) {

*if* (cartItem.product.id === item.product.id) {

          cartItem.amount += item.amount;

        }

      });

*this*.messageService.add('Amount in cart changed for: ' + item.product.name);

    } *else* {

*this*.cartItems.push(item);

*this*.messageService.add('Added to cart: ' + item.product.name);

    }

*this*.itemsChanged.emit(*this*.cartItems.slice());

  }

*public* addItems(items: CartItem[]) {

    items.forEach((cartItem) *=>* {

*this*.addItem(cartItem);

    });

  }

*public* removeItem(item: CartItem) {

*const* indexToRemove = *this*.cartItems.findIndex(element *=>* element === item);

*this*.cartItems.splice(indexToRemove, 1);

*this*.itemsChanged.emit(*this*.cartItems.slice());

*this*.messageService.add('Deleted from cart: ' + item.product.name);

  }

*public* updateItemAmount(item: CartItem, newAmount: number) {

*this*.cartItems.forEach((cartItem) *=>* {

*if* (cartItem.product.id === item.product.id) {

        cartItem.amount = newAmount;

      }

    });

*this*.itemsChanged.emit(*this*.cartItems.slice());

*this*.messageService.add('Updated amount for: ' + item.product.name);

  }

*public* clearCart() {

*this*.cartItems = [];

*this*.itemsChanged.emit(*this*.cartItems.slice());

*this*.messageService.add('Cleared cart');

  }

*public* getTotal() {

*let* total = 0;

*this*.cartItems.forEach((cartItem) *=>* {

      total += cartItem.amount \* cartItem.product.price;

    });

*return* total;

  }

}

Source: https://github.com/satyajitghana/ProjektPlutus/tree/master/src/app/cart

Each of the Order is tied to a Customer, with the following Schema

*export* *class* Customer {

*constructor*(

*public* firstname: string = '',

*public* lastname: string = '',

*public* address1: string = '',

*public* address2: string = '',

*public* zip: number = *null*,

*public* city: string = '',

*public* email: string = '',

*public* phone: string = '',

*public* company: string = '',

*public* country: string = ''

  ) {}

}

And an Order has the following Schema

*export* *class* Order {

*constructor*(

*public* customer: Customer = *null*,

*public* items: CartItem[] = *null*,

*public* total: number = *null*,

*public* status: string = '',

*public* number: string = '',

*public* date: string = *new* Date().toISOString().split('T')[0],

*public* shippingMethod: string = '',

*public* paymentMethod: string = ''

  ) {}

}

Source: https://github.com/satyajitghana/ProjektPlutus/tree/master/src/app/models

We can now look into how Forms are handled, below is the Address Components that manages the address fields during checkout

@Component({

  selector: 'app-checkout-address',

  templateUrl: './address.component.html',

  styleUrls: ['./address.component.scss']

})

*export* *class* AddressComponent *implements* OnInit, OnDestroy {

*private* authSubscription: Subscription;

  @Input() *public* user;

*public* formAddress: FormGroup;

*public* countries: string[];

*constructor*(

*private* checkoutService: CheckoutService,

*private* authService: AuthService

  ) {}

  ngOnInit() {

*this*.initFormGroup();

*this*.authSubscription = *this*.authService.user.subscribe((user) *=>* {

*if* (user) {

*this*.user = user;

*this*.initFormGroup();

      }

    });

  }

*private* initFormGroup() {

*this*.countries = ['India'];

*this*.formAddress = *new* FormGroup({

      firstname: *new* FormControl(

*this*.user && *this*.user.firstName,

        Validators.required

      ),

      lastname: *new* FormControl(

*this*.user && *this*.user.lastName,

        Validators.required

      ),

      address1: *new* FormControl(*null*, Validators.required),

      address2: *new* FormControl(*null*),

      zip: *new* FormControl(*null*, [

        Validators.required,

        Validators.pattern(/*^*\d\d\d\d\d\d*$*/)

      ]),

      city: *new* FormControl(*null*, Validators.required),

      email: *new* FormControl(

*this*.user && *this*.user.email,

        Validators.email

      ),

      phone: *new* FormControl(*null*),

      company: *new* FormControl(*null*),

      country: *new* FormControl({ value: *this*.countries[0], disabled: *false* })

    });

  }

*public* onContinue() {

*this*.checkoutService.setCustomer(*this*.formAddress.value);

*this*.checkoutService.nextStep();

  }

*// Debug: Fill Form Helper MEthod*

*public* onFillForm(event: Event) {

    event.preventDefault();

*this*.formAddress.setValue({

      firstname: 'Satyajit',

      lastname: 'Ghana',

      address1: '14th street, 6th cross',

      address2: 'AECS Layout',

      zip: 560094,

      city: 'Bangalore',

      email: 'shadowleaf.satyajit@gmail.com',

      phone: '+917892137665',

      company: '',

      country: 'India'

    });

  }

  ngOnDestroy() {

*this*.authSubscription.unsubscribe();

  }

}

Source: https://github.com/satyajitghana/ProjektPlutus/blob/master/src/app/checkout/address/

# Bibliography

1. RUAS SDF Lab Manuals, and Assignment of 4th semester, Satyajit Ghana, 17ETCS002159, 2018

2. ProjektPlutus – Satyajit Ghana – SDF Lab final implementation of Online Shopping Website using Angular 6, AngularFirebase and Bootstrap.

3. <https://firebase.google.com/docs/database>

4. https://projekt-plutus.herokuapp.com/