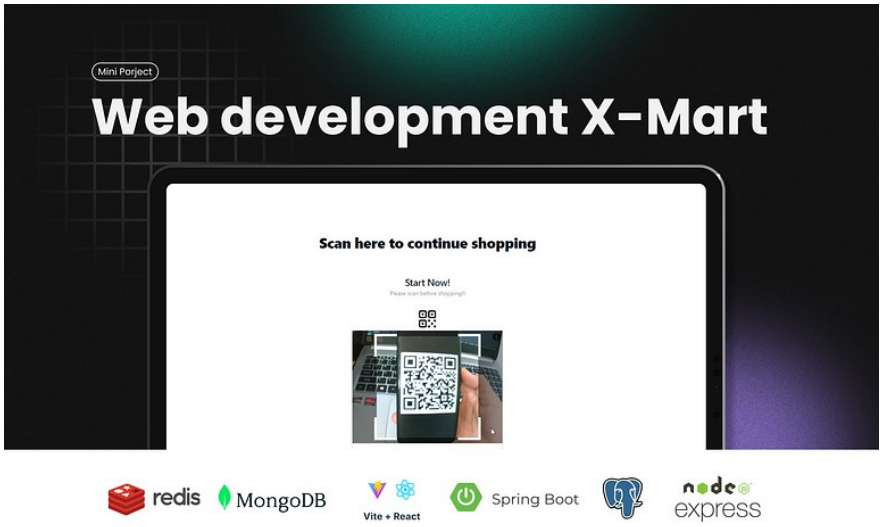
Fullstack Web Development Mini Project X-Mart Menggunakan ReactJS, Java Spring Boot, & Node.js (ExpressJS + GraphQL)

Moch Rizki Kurniawan · Follow
22 min read · May 15, 2024



thumbnail

Halo semuanya

Saya **Moch. Rizki Kurniawan,** seorang software developer. Dalam kesempatan ini, saya ingin berbagi pengalaman tentang pembuatan mini proyek web development X-mart menggunakan teknologi **ReactJS, Java Spring Boot, dan Node.js (ExpressJS + GraphQL)**. Proyek ini juga memanfaatkan **Redis** untuk caching serta mengintegrasikan dua jenis database, yaitu NoSQL **MongoDB** dan SQL **PostgreSQL**.

Apa itu MongoDB,PostgreSQL & Redis?

MongoDB adalah database NoSQL yang sangat fleksibel, memungkinkan penyimpanan data dalam bentuk dokumen JSON. Hal ini memudahkan pengembangan aplikasi yang membutuhkan skema data yang dinamis dan dapat berubah-ubah. Di sisi lain, **PostgreSQL** adalah database SQL yang kuat dan andal, menawarkan fitur-fitur canggih dan kemampuan untuk menangani kueri kompleks.
Redis, digunakan sebagai caching layer, adalah penyimpanan data dalam memori yang sangat cepat. Dengan Redis, kita bisa menyimpan data sementara yang sering diakses, sehingga mengurangi beban pada database utama dan meningkatkan performa aplikasi secara keseluruhan, untuk lebih jelasnya anda bisa melihat pada tautan berikut :

· **mongodb**

· **postgresql**

· **redis**

Konsep Aplikasi

Aplikasi ini mirip dengan platform e-commerce lainnya yang memungkinkan pelanggan untuk melakukan pembelian barang secara online. Untuk memulai, pelanggan dapat menggunakan fitur pemindaian QRCode yang berisi informasi identitas mereka, sehingga terhubung dengan akun wallet mereka dan dapat melakukan pembelian barang secara digital. Pelanggan memiliki opsi untuk menelusuri berbagai barang yang tersedia dan menambahkannya ke keranjang belanja untuk kemudian melakukan proses transaksi pembelian. Setelah transaksi berhasil diselesaikan, pelanggan dapat memeriksa riwayat transaksi sebelumnya untuk memastikan bahwa semua pembelian telah dicatat dengan baik.

1. Persiapan Environtment dan Initialize Project

Sebelum memulai proyek, terdapat beberapa tahapan penting yang harus dilalui dalam menyiapkan *environment* dan menginisialisasi proyek secara efektif. ada beberapa langkah persiapan yang perlu dilakukan sebelumnya diantaranya:

a. Java Spring Boot
Untuk memulai pengembangan aplikasi menggunakan Java Spring Boot, diperlukan Java Development Kit (JDK) dan Maven.

- Silakan unduh dan instal JDK dari situs web resminya: Java Development Kit (JDK).
- Setelah proses instalasi selesai, pastikan JDK terinstal dengan benar dengan menjalankan perintah:

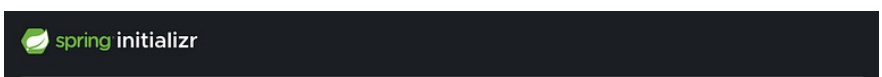
```
java -version
```

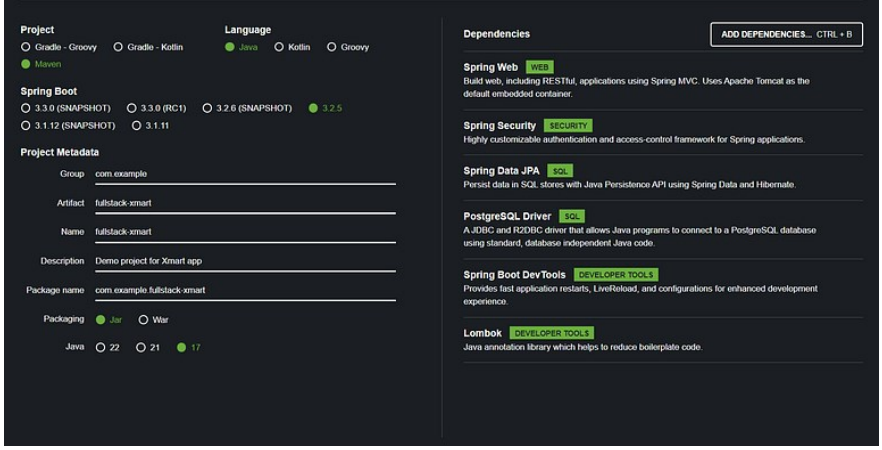
- Selanjutnya, instal Maven dari situs web resminya: Apache Maven.
- Setelah Maven terinstal, periksa versi Maven dengan menjalankan perintah:

```
mvn -version
```

Setelah semuanya terinstal langkah selanjutnya inisiasi *project* springboot dengan maven dapat mengujungi **spring inizializr**

kemudian tambahan *depedencies* sebagai berikut:





spring inizializr

Selanjutnya *generate project* ,dan kemudian buka di texteditor code seperti **vscode** maka tampilan struktur project nya seperti ini:

struktur project awal springboot

b. React JS dengan Vite

- Langkah pertama adalah mengunduh dan menginstal Node.js dari situs web resminya: Node.js.
- Setelah Node.js terinstal, memeriksa versi Node.js dan npm dengan menjalankan perintah berikut di terminal atau command prompt:

```
node -v
npm -v
```

- Pastikan telah menginstal versi Node.js yang kompatibel dengan device yang dimiliki. Versi yang direkomendasikan adalah versi LTS (Long-Term Support).
- Setelah Node.js terinstal, Selanjutnya membuat proyek React menggunakan Vite dengan menjalankan perintah:

```
npm create vite@latest project-xmart  -- --template react
```

Masuk ke direktori proyek yang baru dibuat dengan menjalankan perintah:

```
cd project-xmart
```

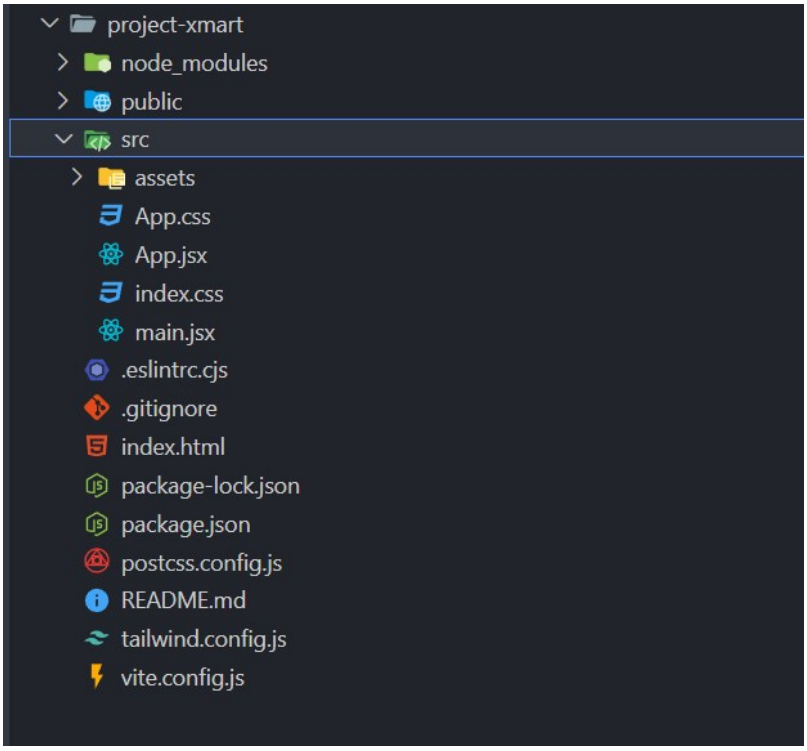
Terakhir, instal semua dependensi yang diperlukan dengan menjalankan perintah:

```
npm install
```

selanjutnya menambahkan tailwindcss untuk implementasi pada project ini dengan perintah:

```
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

Jika berhasil, maka tampilan struktur project nya sebagai berikut :



struktur project awal reacts

c. **Node.js dengan Express.js dan GraphQL**

Untuk mengembangkan server menggunakan Node.js dengan Express.js dan GraphQL, ikuti langkah-langkah berikut:

- Langkah pertama setelah install Node.js sebelumnya, selanjutnya membuat direktori baru untuk proyek nodejs
- Buat direktori baru untuk proyek Anda dan masuk ke dalamnya

```
mkdir project-express-graphql
cd project-express-graphql
```

- Inisialisasi proyek Node.js dengan npm:

```
npm init -y
```

- Instal Express.js, GraphQL, dan Apollo Server dengan integrasi mongodb,postgree dan redis:

```
npm install express graphql express-graphql mongoose cors pg redis
```

d. MongoDB

Untuk penggunaan MongoDB, langkah-langkahnya sebagai berikut:

- Unduh dan instal MongoDB dari situs web resminya: MongoDB.
- Ikuti petunjuk instalasi yang disediakan pada situs web untuk sistem operasi yang Anda gunakan.

e. PostgreSQL

Untuk menggunakan PostgreSQL, lakukan langkah-langkah berikut:

- Unduh dan instal PostgreSQL dari situs web resminya: PostgreSQL.
- Ikuti petunjuk instalasi yang disediakan pada situs web untuk sistem operasi yang Anda gunakan.

f. Redis

Untuk menggunakan Redis, lakukan langkah-langkah berikut:

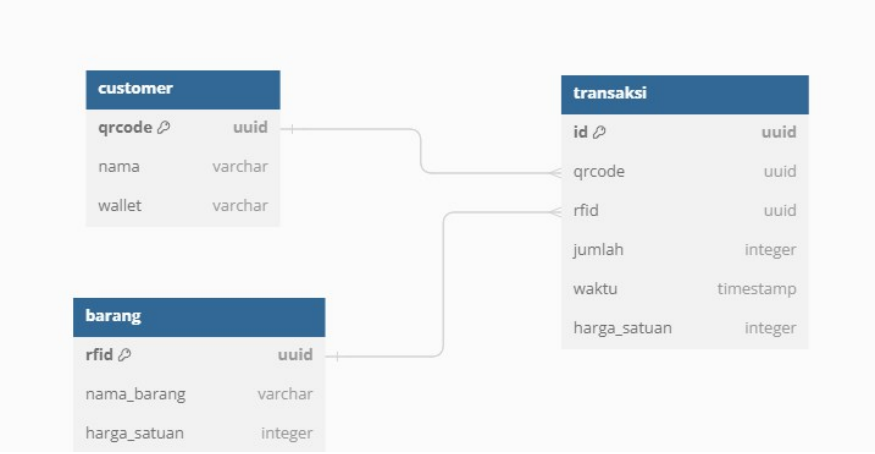
· Unduh dan instal Redis dari situs web resminya: **Redis**

· Ikuti petunjuk instalasi yang disediakan pada situs web untuk sistem operasi yang Anda gunakan.

· Setelah instalasi selesai, jalankan Redis server dengan perintah `redis-server`.

2. Membuat API Springboot

Pada tahap ini, proses pembuatan API menggunakan Spring Boot yang meliputi tiga entitas utama: customer, barang, dan transaksi. API ini akan diintegrasikan dengan PostgreSQL sebagai database utama untuk menyimpan dan mengelola data. untuk schema database nya seperti berikut:



schema database pg

Struktur project springboot keseluruhan:

```
xmart-app
│
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com
│   │   │       └── example
│   │   │           └── xmart
│   │   │               ├── ProjectUsecaseXmartApplication.java
│   │   │               ├── config
│   │   │               │   ├── ApplicationConfig.java
│   │   │               │   ├── ResponseEntityExceptionConfig.java
│   │   │               │   └── WebMvcConfig.java
│   │   │               ├── controller
│   │   │               │   ├── BarangController.java
│   │   │               │   ├── CustomerController.java
│   │   │               │   └── TransaksiController.java
│   │   │               ├── dto
│   │   │               │   ├── request
│   │   │               │   │   ├── AddBarangDto.java
│   │   │               │   │   ├── AddCustomerDto.java
│   │   │               │   │   ├── AddTransaksiDto.java
│   │   │               │   │   ├── EditBarangDto.java
│   │   │               │   │   └── EditCustomerDto.java
│   │   │               │   ├── response
│   │   │               │   │   ├── ResponseBarangDto.java
│   │   │               │   │   ├── ResponseBodyDto.java
│   │   │               │   │   ├── ResponseCustomerDto.java
│   │   │               │   │   └── ResponseTransaksiDto.java
│   │   │               │   └── ErrorDto.java
│   │   │               ├── exception
│   │   │               │   └── MultipleException.java
│   │   │               ├── model
│   │   │               │   ├── Barang.java
│   │   │               │   ├── Customer.java
│   │   │               │   └── Transaksi.java
│   │   │               ├── repository
│   │   │               │   ├── BarangRepository.java
│   │   │               │   ├── CustomerRepository.java
│   │   │               │   └── TransaksiRepository.java
│   │   │               └── services
│   │   │                   ├── BarangService.java
│   │   │                   ├── CustomerService.java
│   │   │                   └── TransaksiService.java
│   │   └── resources
│   │       ├── application.properties
│   │       ├── static
│   │       └── templates
│   │
│   └── test
│       └── java
│           └── com
│               └── example
│                   └── xmart
│                       └── ProjectUsecaseXmartApplicationTests.java
│
├── .gitignore
├── README.md
├── mvnw
├── mvnw.cmd
└── pom.xml
```

Langkah awal adalah menambahkan konfigurasi pada file **application.properties**. Konfigurasi ini sangat penting karena akan menghubungkan aplikasi dengan database PostgreSQL selama proses pengembangan.

```
spring.application.name=project-usecase-xmart

# Database Connection Configuration postgress
spring.datasource.url=jdbc:postgresql://localhost:5432/{database}
spring.datasource.username=username
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver

# Hibernate Properties
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.main.allow-bean-definition-overriding=true
spring.jpa.hibernate.use-new-id-generator-mappings=true
spring.jpa.properties.hibernate.id.new_generator_mappings=true
spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.use_sql_comment=true
spring.jpa.properties.hibernate.format_sql=true
```

Langkah berikutnya, membuat models/entitas seperti berikut :

Barang.java

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@EntityListeners(AuditingEntityListener.class)
@Table(name = "barang")
public class Barang {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "rfid", unique = true, nullable = false)
    private UUID rfid;

    public Barang(UUID rfid) {
        this.rfid = rfid;
    }

    @Column(name = "nama_barang")
    private String namaBarang;

    @Column(name = "harga_satuan")
    private int hargaSatuan;

    @OneToMany(mappedBy = "barang")
    private Set<Transaksi> transaksies;

}
```

Customer.java

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@EntityListeners(AuditingEntityListener.class)
@Table(name = "customer")
public class Customer {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "qrcode", unique = true, nullable = false)
    private UUID qrcode;

    public Customer(UUID qrcode) {
        this.qrcode = qrcode;
    }

    @Column(name = "nama")
    private String nama;

    @Column(name = "wallet")
    private String wallet;

    @OneToMany(mappedBy = "customer")
    private Set<Transaksi> transaksies;

}
```

Transaksi.java

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@EntityListeners(AuditingEntityListener.class)
@Table(name = "transaksi")
public class Transaksi {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", unique = true, nullable = false)
    private UUID id;

    @ManyToOne
    @JoinColumn(name = "qrcode", referencedColumnName = "qrcode")
    private Customer customer;

    @ManyToOne
    @JoinColumn(name = "rfid", referencedColumnName = "rfid")
    private Barang barang;

    @Column(name = "harga_satuan")
    private int hargaSatuan;

    @Column(name = "jumlah")
    private int jumlah;

    @CreatedDate
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "waktu")
    private Timestamp waktu;

}
```

Setelahnya, membuat DTO untuk permintaan (request) dan (response). DTO, atau Data Transfer Object, digunakan untuk mengelola informasi yang dikirim antara komponen/object dalam aplikasi.

Request

AddBarangDto.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AddBarangDTO {
    private String namaBarang;
    private int hargaSatuan;

}
```

AddCustomerDto.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AddCustomerDTO {
    private String nama;
    private String wallet;

}
```

AddTransaksiDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AddTransaksiDTO {
    private UUID qrcode;
    private UUID rfid;
    private int hargaSatuan;
    private int jumlah;
    private Date waktu;
}
```

EditBarangDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class EditBarangDTO {
    private String namaBarang;
    private int hargaSatuan;

}
```

EditCustomerDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class EditCustomerDTO {
    private String nama;
    private String wallet;
}
```

Response

ResponseBarangDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ResponseBarangDTO {
    private UUID rfid;
    private String namaBarang;
    private int hargaSatuan;

}
```

ResponseBodyDTO.java

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class ResponseBodyDTO {
 private long total;
 private Object data;
 private String message;
 private int statusCode;
 private String status;
}
```

ResponseCustomerDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ResponseCustomerDTO {
    private UUID qrcode;
    private String nama;
    private String wallet;

}
```

ResponseTransaksiDTO.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ResponseTransaksiDTO {
    private UUID id;
    private UUID qrcode;
    private UUID rfid;
    private int hargaSatuan;
    private int jumlah;
    private String waktu;
}
```

ErorrDto.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorDTO {
    private int statusCode;
    private String message;
    private String details;
}
```

Selanjutnya disini akan membuat controller nya ,diantaranya :

BarangController.java

```
@RestController
@RequestMapping("/api/")
public class BarangController {

    @Autowired
    private BarangService barangService;
```

```
    @GetMapping("products")
    public ResponseEntity<Object> findAllProduct() {
        return barangService.getAllBarang();
    }

    @GetMapping("products/{id}")
    public ResponseEntity<Object> findByIdProduct(@PathVariable("id") UUID id)
{
        return barangService.getBarangById(id);
    }

    @PostMapping("products/add")
    public ResponseEntity<Object> createProduct(@RequestBody AddBarangDTO addBa
rang) {
        return barangService.addToBarang(addBarang);
    }

    @PutMapping("products/edit/{rfid}")
    public ResponseEntity<Object> editProduct(@PathVariable UUID rfid,
            @RequestBody EditBarangDTO editProducts) {
        return barangService.editBarang(rfid, editProducts);
    }

}
```

CustomerController.java

```
@RestController
@RequestMapping("/api/")
public class CustomerController {

    @Autowired
    private CustomerService customerService;

    @GetMapping("customers")
    public ResponseEntity<Object> findAllCustomer() {
        return customerService.getAllCustomer();
    }

    @GetMapping("customers/{id}")
    public ResponseEntity<Object> findByIdCustomer(@PathVariable("id") UUID id)
 {
        return customerService.getCustomerById(id);
    }

    @PostMapping("customers/add")
    public ResponseEntity<Object> createCustomer(@RequestBody AddCustomerDTO ad
dCustomer) {
        return customerService.addToCustomer(addCustomer);
    }

    @PutMapping("customers/edit/{qrcode}")
    public ResponseEntity<Object> editCustomer(@PathVariable UUID qrcode,
            @RequestBody EditCustomerDTO editCustomer) {
        return customerService.editCustomer(qrcode, editCustomer);
    }


}
```

TransaksiController.java

```
@RestController
@RequestMapping("/api/")
public class TransaksiController {

    @Autowired
    private TransaksiService transaksiService;

    @GetMapping("transactions")
    public ResponseEntity<Object> findAllTransactions() {
        return transaksiService.getAllTransactions();
    }
```

```
    @GetMapping("transactions/{id}")
    public ResponseEntity<Object> findByIdTransactions(@PathVariable("id") UUID
 id) {
        return transaksiService.getTransactionsById(id);
    }

    @PostMapping("transactions/add")
    public ResponseEntity<Object> createTransactions(@RequestBody AddTransaksiD
TO addTransaksi) {
        return transaksiService.addToTransaksi(addTransaksi);
    }

}
```

Selanjutnya untuk, Repository :

BarangRepository.java

```
@Repository
public interface BarangRepository extends JpaRepository<Barang, UUID> {

}
```

CustomerRepository.java

```
@Repository
public interface CustomerRepository extends JpaRepository<Customer, UUID> {

}
```

TransaksiRepository.java

```
@Repository
public interface TransaksiRepository extends JpaRepository<Transaksi, UUID> {

}
```

Kemudian pada bagian service,seperti berikut :

BarangService.java

```
@Service
public class BarangService {

    @Autowired
    private BarangRepository barangRepository;

    @Autowired
    private TransaksiRepository transaksiRepository;

    public ResponseEntity<Object> getAllBarang() {

        HttpStatus status = HttpStatus.OK;

        try {

            List<Barang> barangs = barangRepository.findAll();

            List<ResponseBarangDTO> responseBarangDTOList = barangs.stream()
                    .map(barang -> new ResponseBarangDTO(
                            barang.getRfid(),
                            barang.getNamaBarang(),
                            barang.getHargaSatuan()))
                    .collect(Collectors.toList());

            String message = "Berhasil memuat data";
```

```
            ResponseBodyDTO result = ResponseBodyDTO.builder()
                    .total(barangs.size())
                    .data(responseBarangDTOList)
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();

            return ResponseEntity.ok(result);
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }
    }

    public ResponseEntity<Object> getBarangById(UUID id) {
        HttpStatus status = HttpStatus.OK;

        try {
            Optional<Barang> optionalBarang = barangRepository.findById(id);

            if (optionalBarang.isPresent()) {
                Barang barang = optionalBarang.get();
                ResponseBarangDTO responseBarangDTO = new ResponseBarangDTO(
                        barang.getRfid(),
                        barang.getNamaBarang(),
                        barang.getHargaSatuan());

                String message = "Berhasil memuat data";

                ResponseBodyDTO result = ResponseBodyDTO.builder()
                        .total(1)
                        .data(responseBarangDTO)
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();

                return ResponseEntity.ok(result);
            } else {

                status = HttpStatus.NOT_FOUND;
                String message = "Data tidak ditemukan";
                ResponseBodyDTO notFoundResult = ResponseBodyDTO.builder()
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();
                return ResponseEntity.status(status).body(notFoundResult);
            }
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }
    }

    public ResponseEntity<Object> addToBarang(AddBarangDTO addBarang) {
        HttpStatus status = HttpStatus.CREATED;

        try {

            Barang barang = Barang.builder()
                    .namaBarang(addBarang.getNamaBarang())
                    .hargaSatuan(addBarang.getHargaSatuan())
                    .build();

            barangRepository.save(barang);
            String message = "Berhasil menambahkan barang";

            ResponseBarangDTO responseData = new ResponseBarangDTO(
                    barang.getRfid(),
                    barang.getNamaBarang(),
                    barang.getHargaSatuan());
```

```
            ResponseBodyDTO result = ResponseBodyDTO.builder()
                    .total(1)
                    .data(responseData)
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(result);
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }

    }

    public ResponseEntity<Object> editBarang(UUID rfid, EditBarangDTO editBaran
g) {
        HttpStatus status = HttpStatus.OK;

        try {

            Optional<Barang> optionalBarang = barangRepository.findById(rfid);

            if (optionalBarang.isPresent()) {
                Barang barang = optionalBarang.get();
                barang.setNamaBarang(editBarang.getNamaBarang());
                barang.setHargaSatuan(editBarang.getHargaSatuan());

                barangRepository.save(barang);

                String message = "Berhasil mengedit data customer";

                ResponseBarangDTO responseData = new ResponseBarangDTO(
                        barang.getRfid(),
                        barang.getNamaBarang(),
                        barang.getHargaSatuan());

                ResponseBodyDTO result = ResponseBodyDTO.builder()
                        .total(1)
                        .data(responseData)
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();
                return ResponseEntity.status(status).body(result);
            } else {

                status = HttpStatus.NOT_FOUND;
                String message = "Barang tidak ditemukan";
                ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();
                return ResponseEntity.status(status).body(errorResult);
            }
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }
    }


}
```

CustomerService.java

```
@Service
public class CustomerService {

    @Autowired
    private CustomerRepository customerRepository;
```

```
    public ResponseEntity<Object> getAllCustomer() {

        HttpStatus status = HttpStatus.OK;

        try {
            List<Customer> customers = customerRepository.findAll();

            List<ResponseCustomerDTO> responseCustomerDTOList = customers.strea
m()
                    .map(customer -> new ResponseCustomerDTO(
                            customer.getQrcode(),
                            customer.getNama(),
                            customer.getWallet()))
                    .collect(Collectors.toList());

            String message = "Berhasil memuat data";

            ResponseBodyDTO result = ResponseBodyDTO.builder()
                    .total(customers.size())
                    .data(responseCustomerDTOList)
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();

            return ResponseEntity.ok(result);
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }

    }

    public ResponseEntity<Object> getCustomerById(UUID id) {
        HttpStatus status = HttpStatus.OK;

        try {
            Optional<Customer> optionalCustomer = customerRepository.findById(i
d);

            if (optionalCustomer.isPresent()) {
                Customer customer = optionalCustomer.get();
                ResponseCustomerDTO responseCustomerDTO = new ResponseCustomerD
TO(
                        customer.getQrcode(),
                        customer.getNama(),
                        customer.getWallet());

                String message = "Berhasil memuat data";

                ResponseBodyDTO result = ResponseBodyDTO.builder()
                        .total(1)
                        .data(responseCustomerDTO)
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();

                return ResponseEntity.ok(result);
            } else {

                status = HttpStatus.NOT_FOUND;
                String message = "Data tidak ditemukan";
                ResponseBodyDTO notFoundResult = ResponseBodyDTO.builder()
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();
                return ResponseEntity.status(status).body(notFoundResult);
            }
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }
    }

    public ResponseEntity<Object> addToCustomer(AddCustomerDTO addCustomer) {
        HttpStatus status = HttpStatus.CREATED;
```

```
        try {

            Customer customer = Customer.builder()
                    .nama(addCustomer.getNama())
                    .wallet(addCustomer.getWallet())
                    .build();

            customerRepository.save(customer);

            String message = "Berhasil menambahkan customer";

            ResponseCustomerDTO responseData = new ResponseCustomerDTO(
                    customer.getQrcode(),
                    customer.getNama(),
                    customer.getWallet());

            ResponseBodyDTO result = ResponseBodyDTO.builder()
                    .total(1)
                    .data(responseData)
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(result);
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }

    }

    public ResponseEntity<Object> editCustomer(UUID qrcode, EditCustomerDTO edi
tCustomer) {
        HttpStatus status = HttpStatus.OK;

        try {

            Optional<Customer> optionalCustomer = customerRepository.findById(q
rcode);

            if (optionalCustomer.isPresent()) {
                Customer customer = optionalCustomer.get();
                customer.setNama(editCustomer.getNama());
                customer.setWallet(editCustomer.getWallet());

                customerRepository.save(customer);

                String message = "Berhasil mengedit data customer";

                ResponseCustomerDTO responseData = new ResponseCustomerDTO(
                        customer.getQrcode(),
                        customer.getNama(),
                        customer.getWallet());

                ResponseBodyDTO result = ResponseBodyDTO.builder()
                        .total(1)
                        .data(responseData)
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();
                return ResponseEntity.status(status).body(result);
            } else {

                status = HttpStatus.NOT_FOUND;
                String message = "Customer tidak ditemukan";
                ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();
                return ResponseEntity.status(status).body(errorResult);
            }
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }
    }
```

```
}
```

TransaksiService.java

```
@Service
public class TransaksiService {

    @Autowired
    private TransaksiRepository transaksiRepository;

    public ResponseEntity<Object> getAllTransactions() {

        HttpStatus status = HttpStatus.OK;

        try {
            List<Transaksi> transaksis = transaksiRepository.findAll();
            SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd'T'HH"
, Locale.getDefault());

            List<ResponseTransaksiDTO> responseTransaksiDTOList = transaksis.st
ream()
                    .map(transaksi -> new ResponseTransaksiDTO(
                            transaksi.getId(),
                            transaksi.getCustomer().getQrcode(),
                            transaksi.getBarang().getRfid(),
                            transaksi.getHargaSatuan(),
                            transaksi.getJumlah(),
                            formatter.format(transaksi.getWaktu())))
                    .collect(Collectors.toList());

            String message = "Berhasil memuat data";

            ResponseBodyDTO result = ResponseBodyDTO.builder()
                    .total(transaksis.size())
                    .data(responseTransaksiDTOList)
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();

            return ResponseEntity.ok(result);
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }

    }

    public ResponseEntity<Object> getTransactionsById(UUID id) {
        HttpStatus status = HttpStatus.OK;

        try {
            Optional<Transaksi> optionalTransaksi = transaksiRepository.findByI
d(id);

            if (optionalTransaksi.isPresent()) {
                Transaksi transaksi = optionalTransaksi.get();
                ResponseTransaksiDTO responseTransaksiDTO = convertToResponseDT
O(transaksi);

                String message = "Berhasil memuat data";

                ResponseBodyDTO result = ResponseBodyDTO.builder()
                        .total(1)
                        .data(responseTransaksiDTO)
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();

                return ResponseEntity.ok(result);
            } else {
```

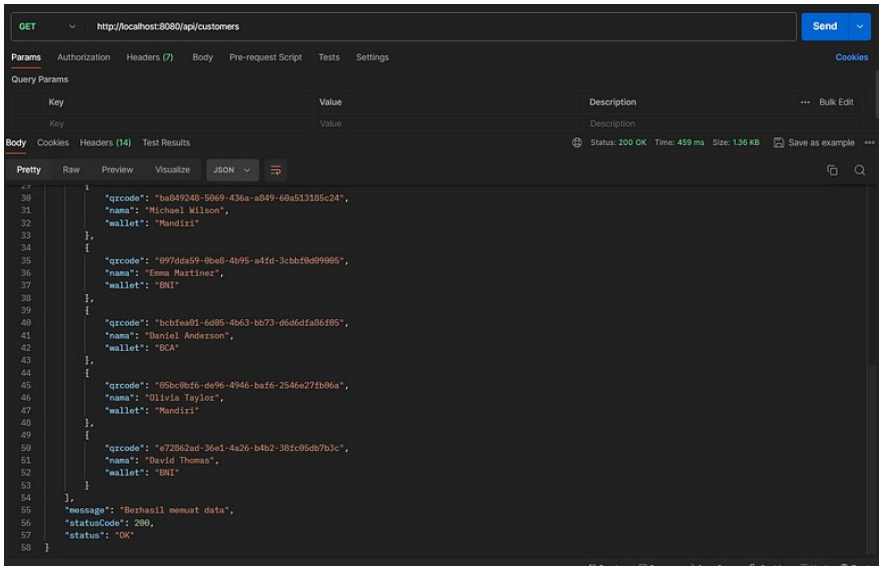
```
                status = HttpStatus.NOT_FOUND;
                String message = "Data tidak ditemukan";
                ResponseBodyDTO notFoundResult = ResponseBodyDTO.builder()
                        .message(message)
                        .statusCode(status.value())
                        .status(status.name())
                        .build();
                return ResponseEntity.status(status).body(notFoundResult);
            }
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }
    }

    public ResponseEntity<Object> addToTransaksi(AddTransaksiDTO addTransaksi)
{
        HttpStatus status = HttpStatus.CREATED;

        try {
            Transaksi transaksi = Transaksi.builder()
                    .customer(new Customer(addTransaksi.getQrcode()))
                    .barang(new Barang(addTransaksi.getRfid()))
                    .hargaSatuan(addTransaksi.getHargaSatuan())
                    .jumlah(addTransaksi.getJumlah())
                    .waktu(new Timestamp(addTransaksi.getWaktu().getTime()))
                    .build();

            transaksiRepository.save(transaksi);

            ResponseTransaksiDTO responseTransaksiDTO = convertToResponseDTO(tr
ansaksi);
            String message = "Berhasil menambahkan transaksi";

            ResponseBodyDTO result = ResponseBodyDTO.builder()
                    .total(1)
                    .data(responseTransaksiDTO)
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(result);
        } catch (Exception e) {
            status = HttpStatus.INTERNAL_SERVER_ERROR;
            String message = "Terjadi kesalahan server";
            ResponseBodyDTO errorResult = ResponseBodyDTO.builder()
                    .message(message)
                    .statusCode(status.value())
                    .status(status.name())
                    .build();
            return ResponseEntity.status(status).body(errorResult);
        }

    }

    private ResponseTransaksiDTO convertToResponseDTO(Transaksi transaksi) {
        ResponseTransaksiDTO responseTransaksiDTO = new ResponseTransaksiDTO();
        responseTransaksiDTO.setId(transaksi.getId());
        responseTransaksiDTO.setQrcode(transaksi.getCustomer().getQrcode());
        responseTransaksiDTO.setRfid(transaksi.getBarang().getRfid());
        responseTransaksiDTO.setHargaSatuan(transaksi.getHargaSatuan());
        responseTransaksiDTO.setJumlah(transaksi.getJumlah());

        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:s
s", Locale.getDefault());
        String formattedDate = formatter.format(transaksi.getWaktu());
        responseTransaksiDTO.setWaktu(formattedDate);

        return responseTransaksiDTO;
    }

}
```

Testing API

Langkah berikutnya adalah menguji API yang telah dibuat menggunakan aplikasi Postman.

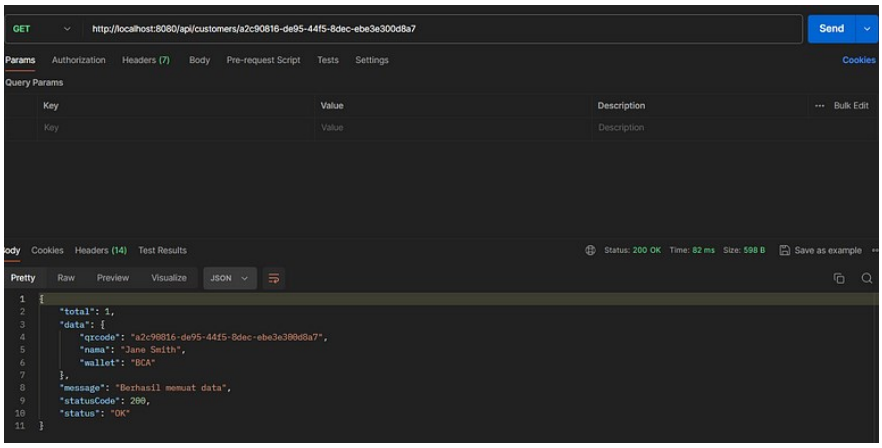
Get all customer

pada bagian ini merupakan Api untuk mendapatkan semua data customer



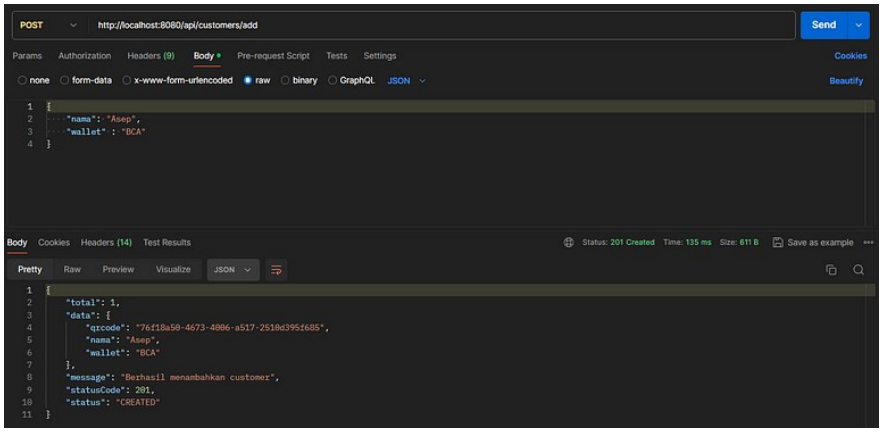
testing api get all customer

Get customer by id



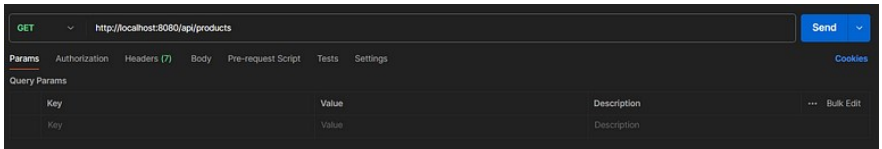
testing api get customer by id

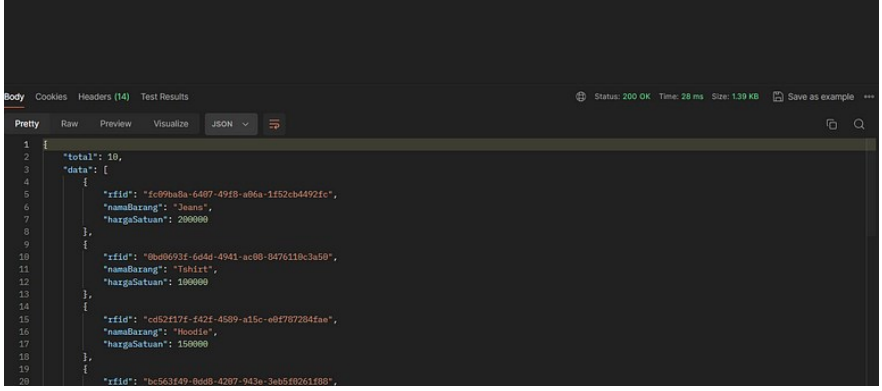
Add customer



testing api add customer

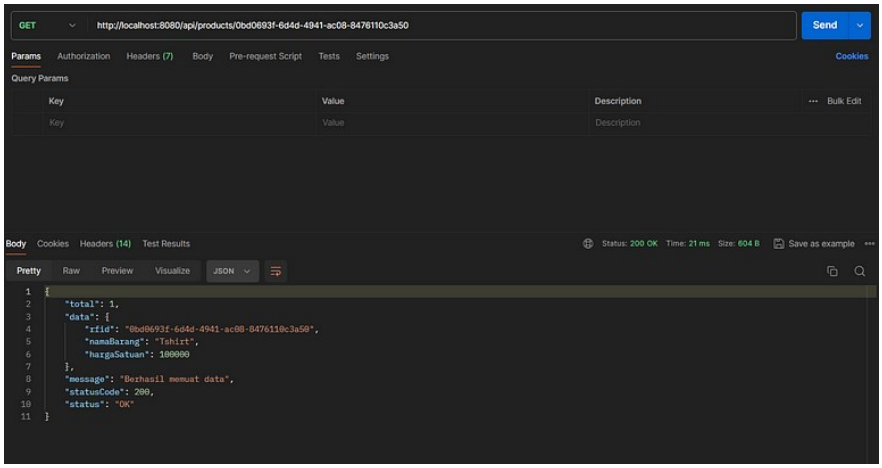
Get All barang





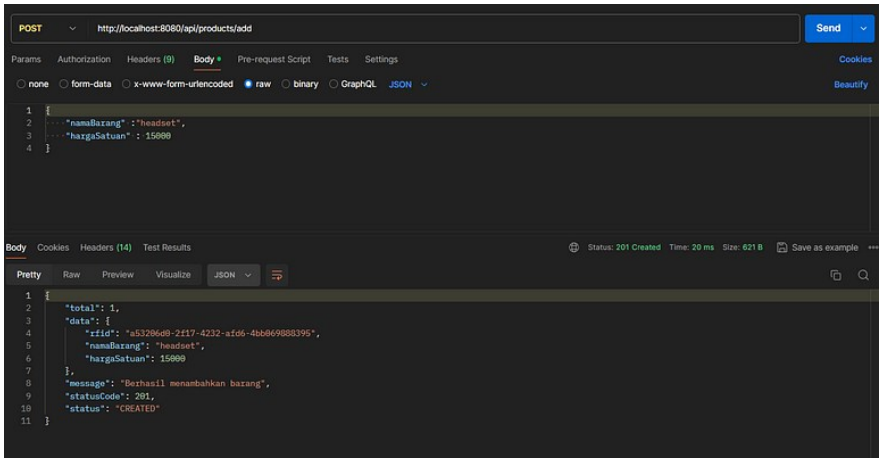
testing api getall barang

Get Barang by id



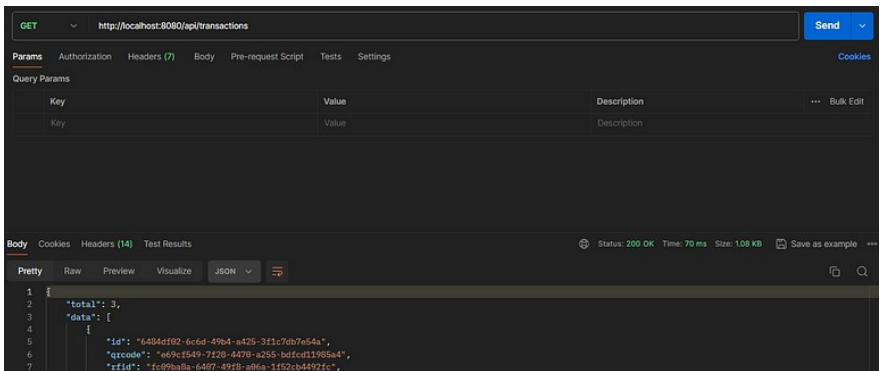
testing api get barang by id

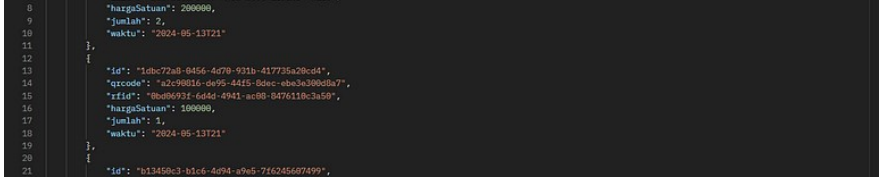
Add Barang



testing api add brang

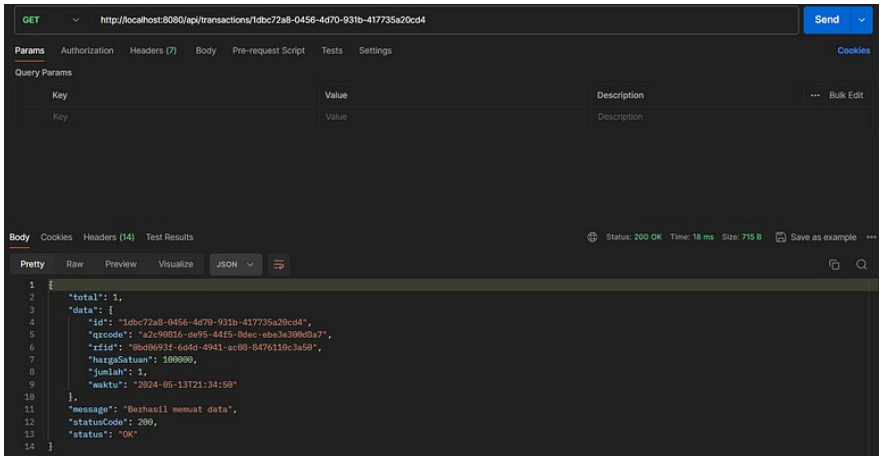
Get All transaksi





testing api get all transaksi

Get transaksi by id



testing api get transaksi by id

3. Membuat API NodeJS (Express -GraphQL)

Pada tahap ini, proses pembuatan API dilakukan menggunakan NodeJS dengan framework Express dan menggunakan GraphQL untuk mengakses dan mengelola data. API ini untuk melakukan berbagai operasi, seperti get data, post data dll, dengan menggunakan query GraphQL . Selain itu, API ini diintegrasikan dengan MongoDB, MySQL, serta Redis. Dalam prosesnya tahapan yang dilakukan pembuatan skema GraphQL yang mendefinisikan tipe data dan operasi yang akan diimplementasiklan, serta resolver yang menghubungkan query dan mutasi GraphQL dengan logika yang sesuai. Langkah awal membuat project pertama dengan perintah

```
mkdir xmart-nodejs
cd xmart-nodejs
npm init -y
```

Selanjutnya membuat project nodejs dengan express redis,postgre,mongo dll , seperti perintah berikut :

```
npm install express graphql express-graphql cors mongoose pg pg-promise pg-hsto
re redis sequelize dotenv
```

Struktur project keseluruhan:

```
xmart-nodejs
├── node_modules
├── src
│   ├── config
│   │   ├── database.js
│   │   ├── redis.js
│   │   └── mongo.js
│   ├── models
│   │   ├── barang.js
│   │   ├── customer.js
│   │   ├── transaksi.js
│   │   ├── transaksiMongo.js
│   │   └── index.js
│   ├── resolvers
│   │   └── index.js
│   ├── schema
│   │   └── index.js
│   ├── test
│   ├── app.js
│   └── index.js
├── .env
├── package-lock.json
└── package.json
```

.env

```
DB_MONGO_URL=mongodb://localhost:27017/database
POSTGRE_USER=username
POSTGRE_PASSWORD=password
POSTGRE_HOST=localhost
POSTGRE_PORT=5432
POSTGRE_DATABASE=database
REDIS_HOST=127.0.0.1
REDIS_PORT=6379
REDIS_DB=0
```

database.js

```
const pgp = require("pg-promise")();

const dbConfig = {
  user: process.env.POSTGRE_USER,
  password: process.env.POSTGRE_PASSWORD,
  host: process.env.POSTGRE_HOST || "localhost",
  port: process.env.POSTGRE_PORT || 5432,
  database: process.env.POSTGRE_DATABASE,
};

const postgreDB = pgp(dbConfig);

postgreDB
  .connect()
  .then((obj) => {
    obj.done();
    console.log("Connection to PostgreSQL has been established successfully.");
  })
  .catch((error) => {
    console.error("Error connecting to PostgreSQL:", error.message);
  });

module.exports = postgreDB;
```

mongo.js

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

```
const mongooseDb = async () => {
  try {
    await mongoose.connect(process.env.DB_MONGO_URL);
    console.log("Connection to MongoDB has been established successfully.");
  } catch (err) {
    console.error("Error connect to MongoDB:", err);
  }
};

module.exports = mongooseDb;
```

redis.js

```
const { createClient } = require("redis");

const REDIS_HOST = process.env.REDIS_HOST || "127.0.0.1";
const REDIS_PORT = process.env.REDIS_PORT || 6379;
const REDIS_DB = process.env.REDIS_DB || 0;

const redisClient = createClient({
  host: REDIS_HOST,
  port: REDIS_PORT,
  db: REDIS_DB,
  tls: false,
});

redisClient.connect().catch(console.error);

redisClient.on("connect", function () {
  console.log("Connection to RedisClient has been established successfully");
});

redisClient.on("error", function (err) {
  console.log("Error connecting to Redis:", err);
});

module.exports = redisClient;
```

barang.js

```
const { DataTypes } = require("sequelize");
const sequelize = require("../config/database");

const Barang = sequelize.define("Barang", {
  rfid: {
    type: DataTypes.UUID,
    defaultValue: DataTypes.UUIDV4,
    primaryKey: true,
  },
  namaBarang: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  hargaSatuan: {
    type: DataTypes.INTEGER,
    allowNull: false,
  },
});

module.exports = Barang;
```

customer.js

```
const { DataTypes } = require("sequelize");
const sequelize = require("../config/database");
```

```
const Customer = sequelize.define("Customer", {
  qrCode: {
    type: DataTypes.UUID,
    defaultValue: DataTypes.UUIDV4,
    primaryKey: true,
  },
  nama: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  wallet: {
    type: DataTypes.STRING,
    allowNull: false,
  },
});

module.exports = Customer;
```

transaksi.js

```
const { DataTypes } = require("sequelize");
const sequelize = require("../config/database");
const Customer = require("./Customer");
const Barang = require("./Barang");

const Transaksi = sequelize.define("Transaksi", {
  id: {
    type: DataTypes.UUID,
    defaultValue: DataTypes.UUIDV4,
    primaryKey: true,
  },
  QRCode: {
    type: DataTypes.UUID,
    references: {
      model: Customer,
      key: "QRCode",
    },
    allowNull: false,
  },
  RFID: {
    type: DataTypes.UUID,
    references: {
      model: Barang,
      key: "RFID",
    },
    allowNull: false,
  },
  Jumlah: {
    type: DataTypes.INTEGER,
    allowNull: false,
  },
  Waktu: {
    type: DataTypes.DATE,
    defaultValue: DataTypes.NOW,
    allowNull: false,
  },
  HargaSatuan: {
    type: DataTypes.INTEGER,
    allowNull: false,
  },
});

Customer.hasMany(Transaksi, { foreignKey: "QRCode" });
Barang.hasMany(Transaksi, { foreignKey: "RFID" });

module.exports = Transaksi;
```

transaksiMongo.js

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

const { Schema } = mongoose;
```

```
const transaksiSchema = new Schema({
  _id: { type: String },
  qrcode: { type: String, required: true },
  rfid: { type: String, required: true },
  harga_satuan: { type: Number, required: true },
  jumlah: { type: Number, required: true },
  date: { type: Date, default: Date.now },
});

const Transaksi = mongoose.model("Transaksi", transaksiSchema);
module.exports = Transaksi;
```

models/index.js

```
const Customer = require("./Customer");
const Barang = require("./Barang");
const Transaksi = require("./Transaksi");
const TransaksiMongo = require("./TransaksiMongo");

module.exports = {
  Customer,
  Barang,
  Transaksi,
  TransaksiMongo,
};
```

resolvers/index.js

```
const Transaksi = require("../models/TransaksiMongo");

const postgreDB = require("../config/database");
const redisClient = require("../config/redis");
const { v4: uuidv4 } = require("uuid");

const root = {
  getTransaksi: async () => {
    try {
      const transactions = await Transaksi.find().limit(100);

      if (!transactions || transactions.length === 0) {
        console.log("No transactions found in the database.");
        return [];
      }

      return transactions;
    } catch (error) {
      console.error("Failed to retrieve transactions:", error);
      throw new Error("Unable to fetch transactions");
    }
  },

  addBarang: async ({ rfid }) => {
    try {
      const cacheData = await redisClient.get(rfid);

      if (cacheData) {
        return JSON.parse(cacheData);
      } else {
        const barang = await postgreDB.oneOrNone(
          "SELECT * FROM barang WHERE rfid = $1",
          [rfid]
        );

        if (barang) {
          await redisClient.setEx(rfid, 120, JSON.stringify(barang));

          return barang;
        } else {
          await redisClient.setEx(rfid, 120, "Try again later");
          throw new Error("Item not found");
        }
      }
    } catch (error) {
      throw new Error(`Error occurred: ${error.message}`);
    }
  },
```

```
  simpanTransaksi: async ({ data: { qrcode, rfid, harga_satuan, jumlah } }) =>
{
    try {
      const currentTime = new Date();

      const transactionData = {
        _id: uuidv4(),
        qrcode,
        rfid,
        harga_satuan,
        jumlah,
        date: currentTime.toISOString(),
      };

      const transaction = new Transaksi(transactionData);
      await transaction.save();
      console.log(
        "Transaction data has been saved to MongoDB:",
        transactionData
      );
      return transactionData;
    } catch (error) {
      console.error("Failed to save transaction data to MongoDB:", error);
      throw new Error("Failed to save transaction data to MongoDB");
    }
  },
};

module.exports = root;
```

schema/index.js

```
const { buildSchema } = require("graphql");

const schema = buildSchema(`
  type Barang {
    rfid: String!
    nama_barang: String!
    harga_satuan: Int!
  }

  type Transaksi {
    _id: String!
    qrcode: String!
    rfid: String!
    harga_satuan: Int!
    jumlah: Int!
    date: String!
  }

  input TransaksiQuery {
    qrcode: String!
    rfid: String!
    harga_satuan: Int!
    jumlah: Int!
  }

  type Query {
    addBarang(rfid: String!): Barang
    getTransaksi: [Transaksi]!
  }

  type Mutation {
    simpanTransaksi(data: TransaksiQuery!): Transaksi
  }
`);

module.exports = { schema };
```

app.js

```
const express = require("express");
const { ApolloServer } = require("apollo-server-express");
const connectMongoDB = require("./config/mongo");
const { graphqlHTTP } = require("express-graphql");
const cors = require("cors");
const { schema } = require("./schema");
const root = require("./resolvers");
const app = express();

connectMongoDB();

app.use(express.json());
app.use(cors());

app.use(
  "/graphql",
  graphqlHTTP({
    schema: schema,
    rootValue: root,
    graphiql: true,
  })
);
const server = new ApolloServer({
  schema,
});

async function startApolloServer() {
  await server.start();
  server.applyMiddleware({ app });
}

startApolloServer();

module.exports = app;
```

index.js

```
require("dotenv").config();
const app = require("./app");

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

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}/graphql`);
});
```

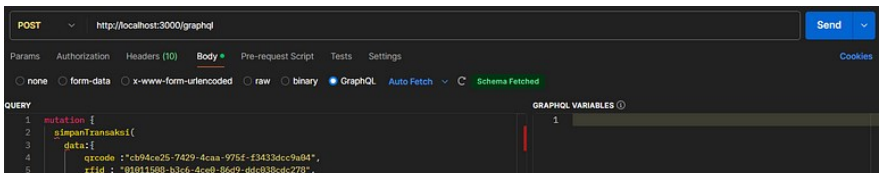
Selanjutnya menjalankan project , jika berhasil maka akan muncul di console seperti ini :

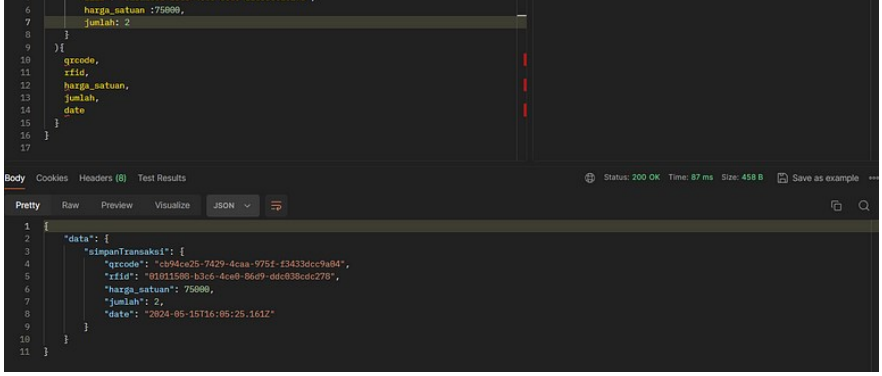
```
> fullstack-xmart-nodejs@1.0.0 start
> node src/index.js

Server is running on http://localhost:3000/graphql
Connection to RedisClient has been established successfully
Connection to MongoDB has been established successfully.
Connection to PostgreSQL has been established successfully.
```

Testing API

Langkah selanjutnya setelah membangun API adalah melakukan pengujian untuk memastikan bahwa API berfungsi dengan baik.

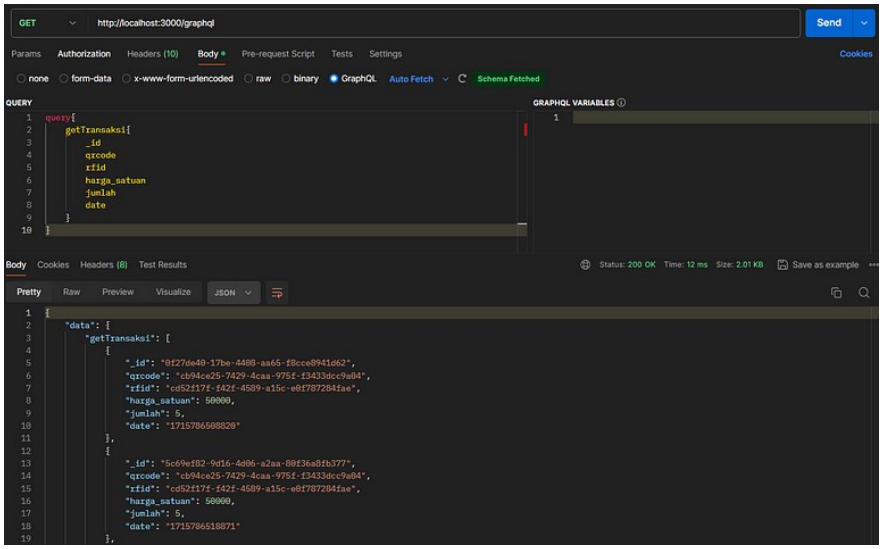
SimpanTransaksi





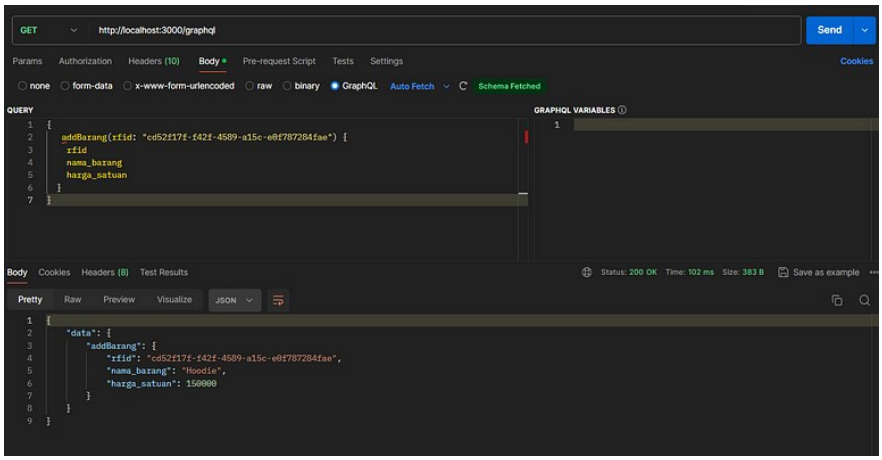
testing api simpan transaksi

GetTransaksi



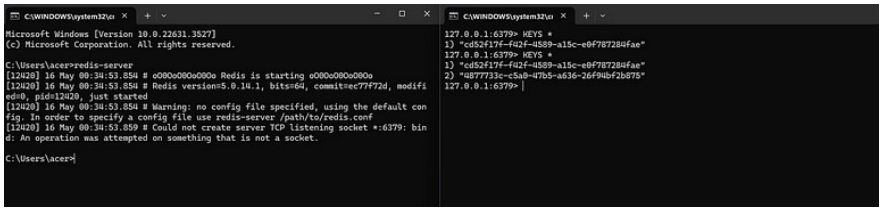
testing api get transaksi

AddBarang



testing api add barang

Pada tahapan ini merupakan proses checkIn barang ke redis, dimana nantinya data ini akan terspimpan ke redis untuk melakukan pengecekan seperti hasil berikut :



redis data

4. Membuat Frontend Reacts + vite

Pada tahap ini, menggunakan React + Vite untuk menghasilkan *user interface*(UI) yang responsif dan dinamis. Selain itu project ini menggunakan tailwindcss, dalam prosesnya membuat berbagai komponen React yang mewakili berbagai bagian dari *user interface*(UI), selain itu melakukan routing untuk navigasi ,dan manajemen state.

Struktur project keseluruhan

```
xmart-reacts/
│
├── public/
│   ├── index.html
│   └── favicon.ico
│
├── src/
│   ├── assets/
│   │   └── img/
│   │
│   ├── components/
│   │   ├── demo/
│   │   │   ├── Data.tsx
│   │   │   ├── DetailTransaction.tsx
│   │   │   ├── History.tsx
│   │   │   ├── ItemList.tsx
│   │   │   └── Sidebar.tsx
│   │   │
│   │   └── ui/
│   │       ├── alert.tsx
│   │       ├── badge.tsx
│   │       ├── button.tsx
│   │       ├── card.tsx
│   │       ├── progress.tsx
│   │       ├── resizable.tsx
│   │       ├── table.tsx
│   │       └── tabs.tsx
│   │
│   ├── pages/
│   │   ├── content/
│   │   └── maincontent/
│   │
│   ├── Routes/
│   │   └── _routes.jsx
│   │
│   ├── App.jsx
│   └── index.jsx
│
├── .gitignore
├── package.json
├── README.md
├── tailwind.config.js
└── vite.config.js
```

Berikut beberapa contoh code component/screen page yang digunakan dalam implementasi menggunakan reacts+vite.

Data.tsx

```
export default function Data() {
  const [customers, setCustomers] = useState([]);
  const [products, setProducts] = useState([]);
  const [transaction, setTransaction] = useState([]);

  function formatRupiah(angka) {
    let number_string = angka.toString();
    let sisa = number_string.length % 3;
    let rupiah = number_string.substr(0, sisa);
    let ribuan = number_string.substr(sisa).match(/\d{3}/g);
```

```
  if (ribuan) {
    let separator = sisa ? "." : "";
    rupiah += separator + ribuan.join(".");
  }

  return "Rp " + rupiah;
}

useEffect(() => {
  axios
    .get("http://localhost:8080/api/customers")
    .then((response) => {
      setCustomers(response.data.data);
      console.log(response.data.data);
    })
    .catch((error) => {
      console.error("Error fetching customer lists:", error);
    });
}, []);

useEffect(() => {
  axios
    .get("http://localhost:8080/api/products")
    .then((response) => {
      setProducts(response.data.data);
      console.log(response.data.data);
    })
    .catch((error) => {
      console.error("Error fetching products lists:", error);
    });
}, []);
useEffect(() => {
  axios
    .get("http://localhost:8080/api/transactions")
    .then((response) => {
      setTransaction(response.data.data);
      console.log(response.data.data);
    })
    .catch((error) => {
      console.error("Error fetching products lists:", error);
    });
}, []);
```

```
  return (
    <div className="bg-neutral-100">
      <div className="flex flex-row w-full h-full mx-auto gap-5 p-5 ">
        {/* Sidebar */}
        <div className="h-[calc(100vh-2rem)] w-full max-w-[20rem] ">
          <Sidebar />
        </div>
        <div className="flex flex-col w-full h-full items-center justify-center
 mx-auto">
          <div className="flex flex-row w-full h-full gap-3 p-5 m-1 rounded-xl
shadow-blue-gray-900/5 bg-slate-700 ">
            <a href="#">
              <MagnifyingGlassIcon className="text-white w-5 h-5" />
            </a>
            <a href="#">
              <ArrowsPointingInIcon className="text-white w-5 h-5" />
            </a>
            <a href="#">
              <BellIcon className="text-white w-5 h-5" />
            </a>
          </div>
        <div className="flex flex-row w-full h-full items-center justify-center
 mx-auto mt-14 ">
          <div className=" rounded-2xl w-1/1 p-5 overflow-auto ">
            <div className="mx-auto max-w-7xl px-4 py-6 sm:px-6 lg:px-8">
              <h1 className="text-3xl font-bold tracking-tight text-gray-900 te
xt-center">
                Data
              </h1>
            </div>
            <div className="mx-auto w-[70rem] max-w-[70rem] py-6 sm:px-6 lg:px-
8 ">
              <div className="bg-white">
                <Tabs defaultValue="customer" className="max-w-full">
                  <TabsList className="grid w-full grid-cols-3 bg-gray-200 ">
                    <TabsTrigger value="customer">Customer</TabsTrigger>
                    <TabsTrigger value="product">Product</TabsTrigger>
                    <TabsTrigger value="transaction">Transaction</TabsTrigger>
                  </TabsList>
                  <TabsContent value="customer" className="pt-6">
                    <Card>
                      <CardHeader>
                        <CardTitle>Customer Data</CardTitle>
                      </CardHeader>
                      <CardContent className="space-y-2">
                        <div className="bg-white">
                          <Table>
                            <TableHeader>
                              <TableRow>
                                <TableHead>Qrcode</TableHead>
                                <TableHead>Name</TableHead>
                                <TableHead className="text-right">
                                  Wallet
                                </TableHead>
                              </TableRow>
                            </TableHeader>
                            <TableBody>
                              {customers.map((customer, index) => (
                                <TableRow
                                  key={customer.qrcode}
                                  className={
                                    index % 2 === 0 ? "bg-blue-100" : ""
                                  }
                                >
                                  <TableCell className="font-medium">
                                    {customer.qrcode}
                                  </TableCell>
                                  <TableCell>{customer.nama}</TableCell>
                                  <TableCell className="text-right">
                                    {customer.wallet}
                                  </TableCell>
                                </TableRow>
                              ))}
                            </TableBody>
                          </Table>
                        </div>
                      </CardContent>
                      <CardFooter></CardFooter>
                    </Card>
                  </TabsContent>
                  <TabsContent value="product" className="pt-6">
                    <Card>
                      <CardHeader>
                        <CardTitle>Products Data</CardTitle>
                      </CardHeader>
                      <CardContent className="space-y-2">
                        <div className="bg-white">
                          <Table>
                            <TableHeader>
                              <TableRow>
                                <TableHead>Rfid</TableHead>
                                <TableHead>Nama Barang</TableHead>
```

```
                            <TableHead className="text-right">
                              Harga satuan
                            </TableHead>
                          </TableRow>
                        </TableHeader>
                        <TableBody>
                          {products.map((products, index) => (
                            <TableRow
                              key={products.rfid}
                              className={
                                index % 2 === 0 ? "bg-blue-100" : ""
                              }
                            >
                              <TableCell className="font-medium">
                                {products.rfid}
                              </TableCell>
                              <TableCell>{products.namaBarang}</TableCell>
                              <TableCell className="text-right">
                                {formatRupiah(products.hargaSatuan)}
                              </TableCell>
                            </TableRow>
                          ))}
                        </TableBody>
                      </Table>
                    </div>
                  </CardContent>
                  <CardFooter></CardFooter>
                </Card>
              </TabsContent>
              <TabsContent value="transaction" className="pt-6">
                <Card>
                  <CardHeader>
                    <CardTitle>Transaction Data</CardTitle>
                  </CardHeader>
                  <CardContent className="space-y-2">
                    <div className="bg-white">
                      <Table>
                        <TableHeader>
                          <TableRow>
                            <TableHead>Qrcode</TableHead>
                            <TableHead>Rfid</TableHead>
                            <TableHead>Harga satuan</TableHead>
                            <TableHead>Jumlah</TableHead>
                            <TableHead className="text-right">
                              Waktu
                            </TableHead>
                          </TableRow>
                        </TableHeader>
                        <TableBody>
                          {transaction.map((transaction, index) => (
                            <TableRow
                              key={transaction.id}
                              className={
                                index % 2 === 0 ? "bg-blue-100" : ""
                              }
                            >
                              <TableCell className="font-medium">
                                {transaction.qrcode}
                              </TableCell>
                              <TableCell>{transaction.rfid}</TableCell>
                              <TableCell>
                                {formatRupiah(transaction.hargaSatuan)}
                              </TableCell>
                              <TableCell>{transaction.jumlah}</TableCell>
                              <TableCell className="text-right">
                                {transaction.waktu}
                              </TableCell>
                            </TableRow>
                          ))}
                        </TableBody>
                      </Table>
                    </div>
                  </CardContent>
                  <CardFooter></CardFooter>
                </Card>
              </TabsContent>
            </Tabs>
          </div>
        </div>
      </div>
    </div>
  </div>
);
```

MainContent

```
function MainContent() {
  const [showScanner, setShowScanner] = useState(true);
  const [showLoading, setShowLoading] = useState(false);
  const [showStart, setShowStart] = useState(false);
  const [progress, setProgress] = React.useState(2);
  const navigate = useNavigate();
  const [getClient, setClient] = useState(null);

  const toggleScanner = () => {
    setShowLoading(true);

    setTimeout(() => {
      setShowScanner(!showScanner);
      setShowLoading(false);
    }, 2000);
  };

  React.useEffect(() => {
    const timer = setTimeout(() => setProgress(66), 2000);
    return () => clearTimeout(timer);
  }, []);

  const handleStartShopping = () => {
    setShowLoading(true);
    setShowStart(true);

    setTimeout(() => {
      setShowScanner(!showScanner);
      setShowLoading(false);
      navigate("/shop");
    }, 2000);
  };

  useEffect(() => {
    axios
      .get(
        "http://localhost:8080/api/customers/{$qrcode}"
      )
      .then((response) => {
        setClient(response.data.data);
        console.log(response.data.data);
      })
      .catch((error) => {
        console.error("Error fetching item lists:", error);
      });
  }, []);

  const [result, setResult] = useState("No result");
  const qrCodeScannerRef = useRef(null);

  useEffect(() => {
    if (qrCodeScannerRef.current) {
      const scanner = new Html5QrcodeScanner(
        "qr-reader",
        { fps: 20, qrbox: 200 },
        false
      );

      scanner.render(onScanSuccess, onScanError);

      return () => {
        scanner.clear().catch((error) => {
          console.error("Failed to clear scanner. ", error);
        });
      };
    }
  }, []);

  const onScanSuccess = (decodedText, decodedResult) => {
    if (decodedText === "{$qrcode}") {
      toggleScanner();
    } else {
      setResult(decodedText);
    }
  };

  const onScanError = (errorMessage) => {
    console.error(errorMessage);
  };
```

```
  return (
    <>
      <>
        <div className="flex items-center justify-center h-screen">
          <div className="text-center">
            {showScanner ? (
              <>
                <h1 className="scroll-m-20 text-4xl font-extrabold tracking-tig
ht lg:text-4xl py-10">
                  Scan here to continue shopping
                </h1>
                <Card
                  className="w-[450px]"
                  style={{ backgroundColor: "#fbfbfb" }}
                >
                  <CardHeader>
                    <CardTitle>Start Now!</CardTitle>
                    <CardDescription>
                      Please scan before shopping!!
                    </CardDescription>
                  </CardHeader>
                  <CardContent>
                    <div className="flex items-center justify-center ">
                      <QrCodeIcon className="w-14 h-14 text-black" />
                    </div>
                    <div className="flex items-center justify-center ">
                      <div
                        id="qr-reader"
                        ref={qrCodeScannerRef}
                        style={{ width: "300px" }}
                      ></div>
                    </div>
                  </CardContent>
                  <CardFooter className="flex justify-center">
                    <Button>{result}</Button>
                  </CardFooter>
                </Card>
              </>
            ) : (
              <>
                <Card className="w-[450px]">
                  <CardHeader>
                    <CardTitle>Information</CardTitle>
                    <CardDescription>
                      Detail information customer"
                    </CardDescription>
                  </CardHeader>
                  <CardContent>
                    {getClient && (
                      <div className="flex items-center">
                        <div className="flex-none">
                          <QRCode
                            value={getClient.qrcode}
                            className="object-cover w-[150px] h-[150px] rounded
-md"
                          />
                        </div>
                        <div className="flex-grow ml-4">
                          <h4 className="scroll-m-20 text-xl font-semibold trac
king-tight">
                            {getClient.nama}
                          </h4>
                          <p className="leading-7 [&:not(:first-child)]:mt-6">
                            {getClient.qrcode}
                          </p>
                          <Badge>{getClient.wallet}</Badge>
                        </div>
                      </div>
                    )}
                  </CardContent>
                  <CardFooter className="flex justify-center">
                    <Button onClick={handleStartShopping}>
                      Start Shopping
                    </Button>
                  </CardFooter>
                </Card>
              </>
            )}
            {showLoading && (
              <div>
                <Progress value={progress} className="w-[100%] h-[5px]" />
              </div>
            )}
```

```
            {showStart && (
              <div className="mt-2">
                <Alert className="bg-gray-200">
                  <AlertTitle>Scan Process</AlertTitle>
                  <AlertDescription>
                    The QR code is currently being scanned to facilitate the
                    shopping process.
                  </AlertDescription>
                </Alert>
              </div>
            )}
          </div>
        </div>
      </>
    </>
  );
}

export default MainContent;
```

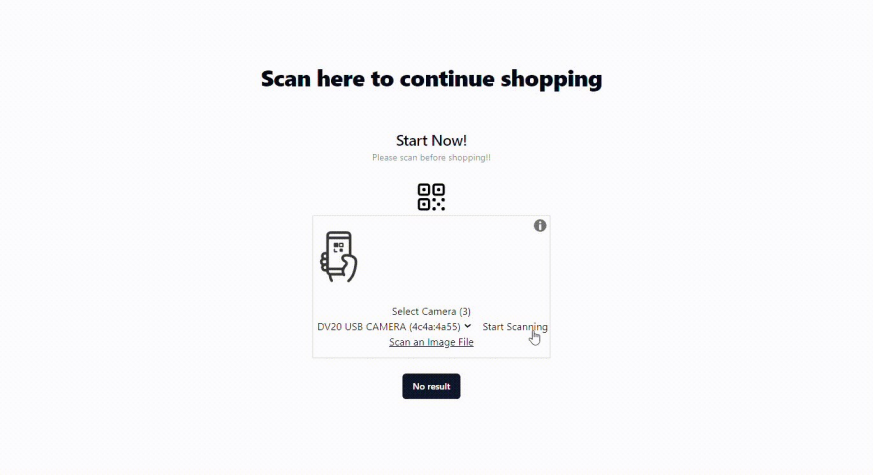
Disini saya hanya mencontohkan beberapa code pada project, namun untuk lebih jelas dan lebih detail anda dapat melihat sumber code pada tautan berikut: **link code**

DEMO

Selanjutnya akan melakukan demo hasil screen yang telah diintegrasikan dengan API yang telah dibuat

1. Screen Landing page

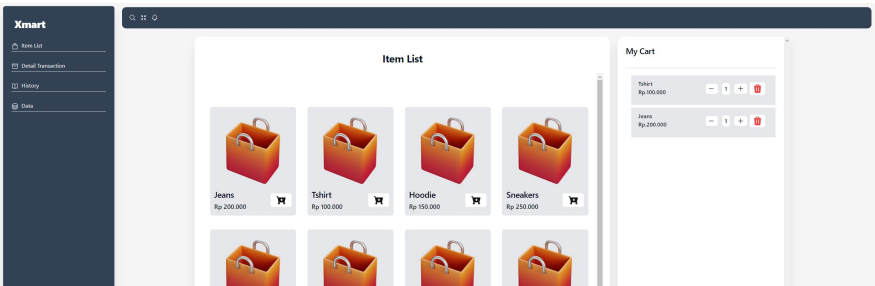
Pada bagian ini merupakan tampilan awal screen yang memuat informasi untuk melakukan barcode scanner bagi user

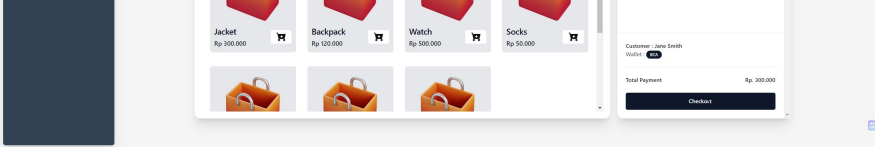


landing page barcode scanner

2. Screen Item list

Pada bagian ini merupakan tampilan screen saat setelah berhasil melakukan barcode scanner dimana memuat informasi daftar barang yang ada kemudian terdapat keranjang saya untuk menambahkan item barang yang ditambahkan

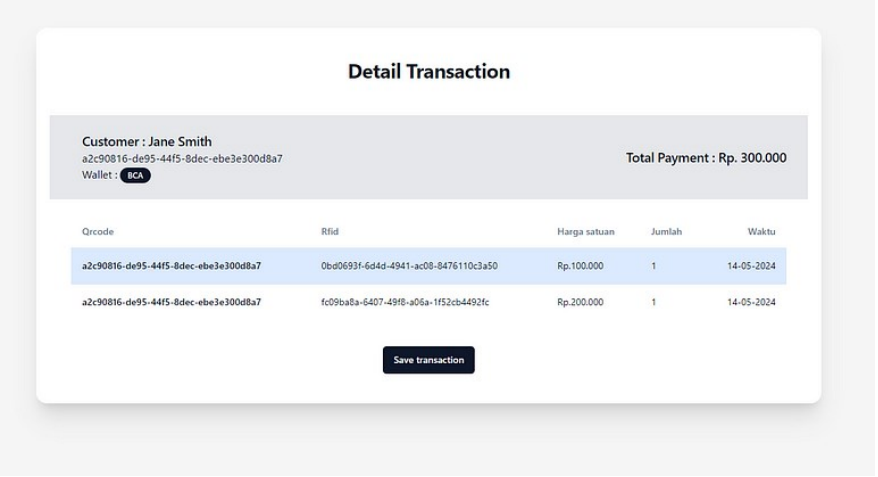




item list

3. Screen Detail transaction

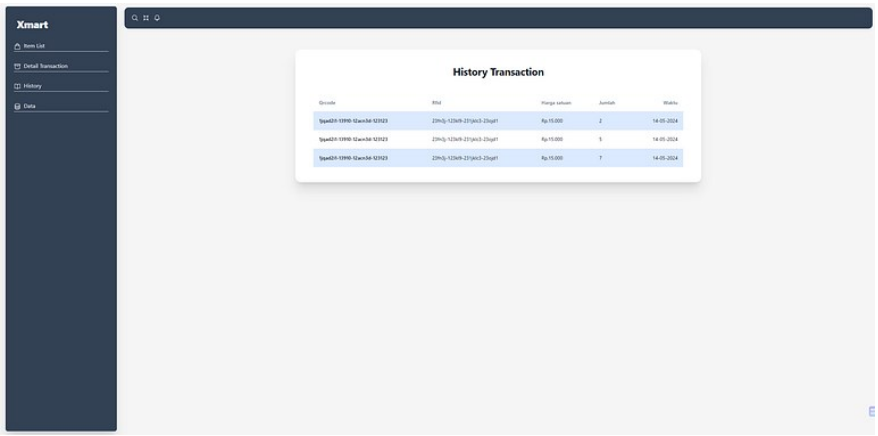
Pada bagian ini merupakan tampilan screen yang memuat informasi data user dan data transaksi barang yang telah dilakukan



detail transaction

4. Screen History transaction

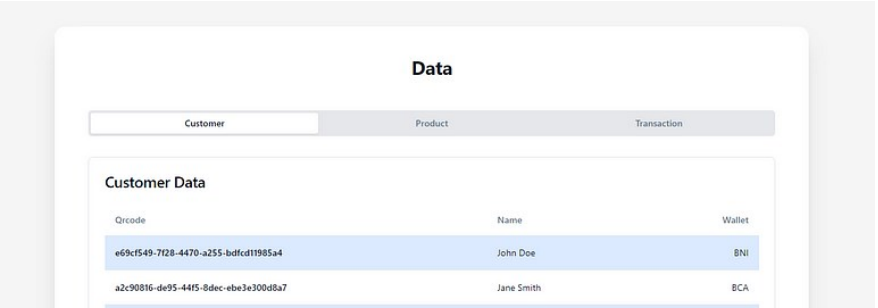
Pada bagian ini merupakan tampilan screen yang memuat kumpulan informasi data histori transaksi yang telah dilakukan



history transaction

5. Screen Data

Pada bagian ini, terdapat tampilan screen yang menampilkan tiga jenis informasi utama: data pelanggan, data barang, dan data transaksi.



cb94ce25-7429-4caa-975f-f3433dcc9a04	Alice Johnson	Mandiri
56060e1b-3425-41b4-820e-d83aeb215f41	Bob Brown	BNI
f3dfd33f-7b7d-4448-a127-a573ad2e8baa	Emily Davis	BCA
ba849248-5069-436a-a849-60a513185c24	Michael Wilson	Mandiri
097dda59-0be8-4b95-a4fd-3cbbf0d09005	Emma Martinez	BNI
bcbfea01-6d05-4b63-bb73-d6d6dfa86f05	Daniel Anderson	BCA
05bc0bf6-de96-4946-baf6-2546e27fb06a	Olivia Taylor	Mandiri
e72862ad-36e1-4a26-b4b2-38fc05db7b3c	David Thomas	BNI
76f18a50-4673-4006-a517-2510d395f685	Asep	BCA

data

Dengan demikian, artikel ini telah memberikan gambaran tentang pembuatan proyek mini X-mart dalampengembangan web. Proyek ini menggunakan teknologi ReactJS, Java Spring Boot, dan Node.js (ExpressJS + GraphQL), serta mengintegrasikan dua jenis database, yaitu NoSQL MongoDB dan SQL PostgreSQL, bersama dengan caching Redis. Untuk melihat proyek lengkapnya, Anda bisa mengakses kode sumber melalui tautan berikut: **[link code].**

Semoga artikel ini memberikan wawasan yang berguna bagi pembaca. Terima kasih atas perhatiannya dan sampai jumpa di artikel berikutnya.

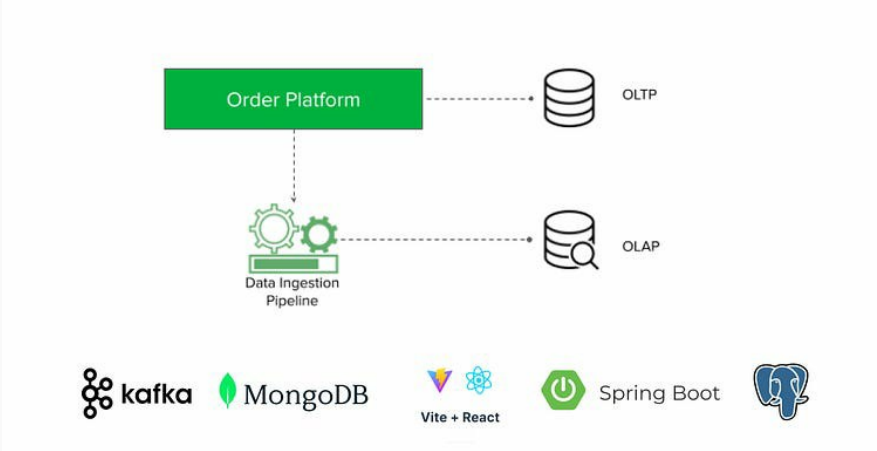
Semangat!

Written by Moch Rizki Kurniawan

0 Followers

Follow

More from Moch Rizki Kurniawan



Moch Rizki Kurniawan

Develop Order Store dengan OLTP , OLAP & Kafka(Use Case Order Grab)

Halo semuanya

24 min read · Apr 18, 2024

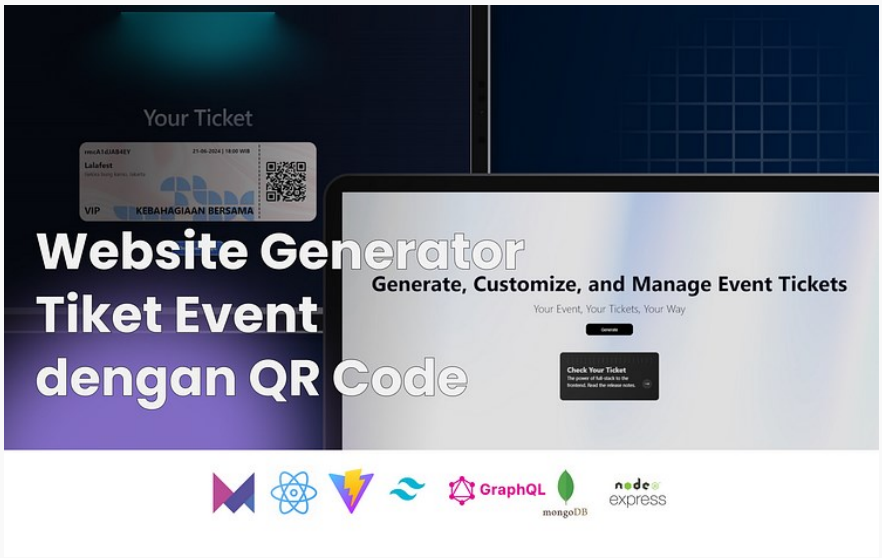
50

Moch Rizki Kurniawan

UI/UX Case Study : My Pets

Halo semuanya

9 min read · Feb 24, 2023

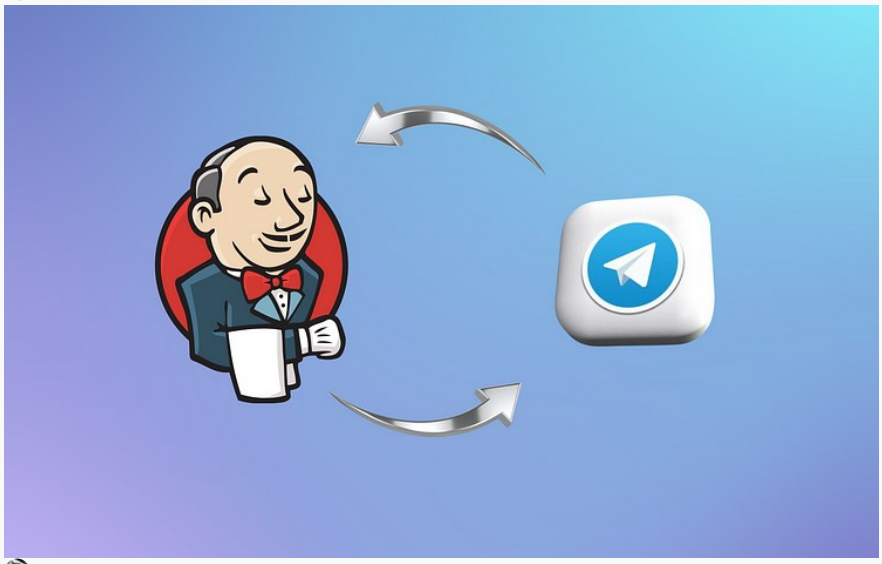


Moch Rizki Kurniawan

Implementasi Website Generator Tiket Event dengan QR Code menggunakan React, Vite, Node.js

Halo semuanya

17 min read · 1 day ago



Moch Rizki Kurniawan

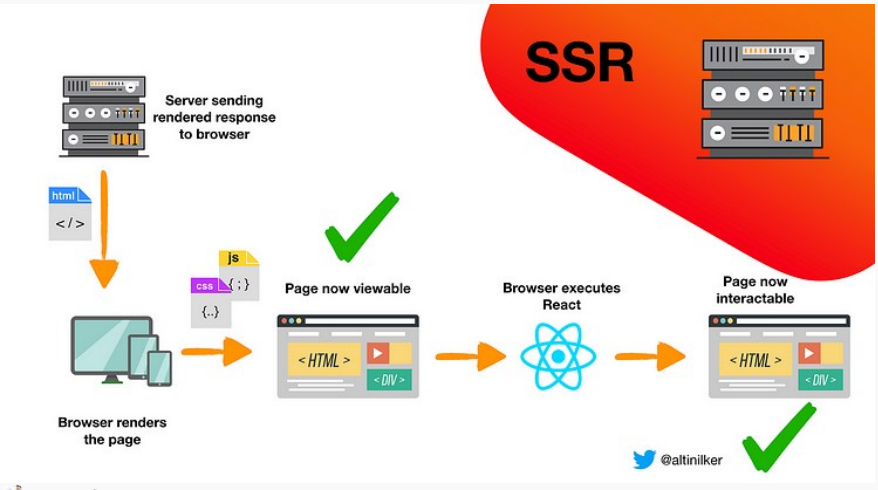
Integrasi Jenkins dengan telegram

Halo semuanya

4 min read · May 21, 2024

See all from Moch Rizki Kurniawan

Recommended from Medium



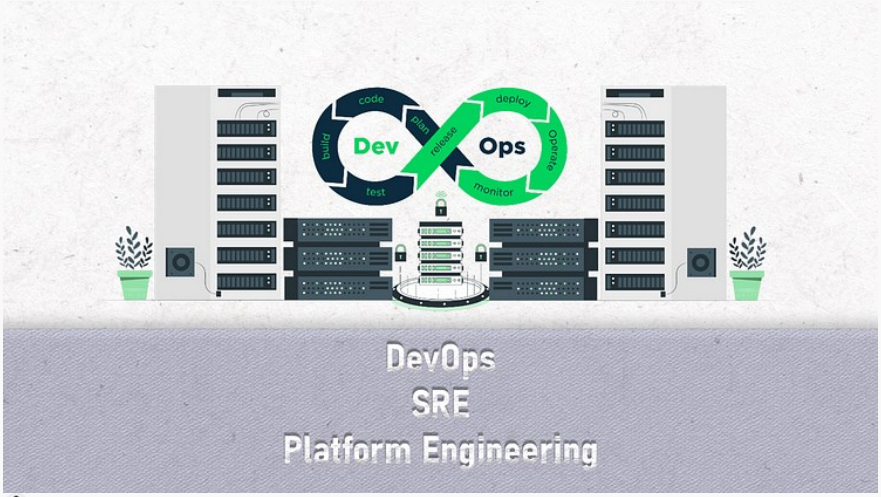
Rishabh Gupta

Understanding Server-Side Rendering (SSR) in React with Vite

Introduction: In the dynamic world of web development, speed and search engine optimization (SEO) are paramount. As developers, we strive...

10 min read · Apr 19, 2024

4



Arton Demaku

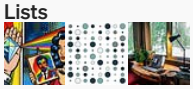
DevOps vs Site Reliability Engineering vs Platform Engineering

In this article we will talk about DevOps, SRE, and Platform Engineering. These concepts might seem confusing at first, but by the end of...

· 3 min read · 6 days ago

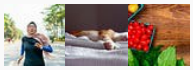
301 6

Lists



Staff Picks
651 stories · 999 saves

Stories to Help You Level-Up at Work
19 stories · 623 saves



Self-Improvement 101
20 stories · 1911 saves

Productivity 101
20 stories · 1744 saves

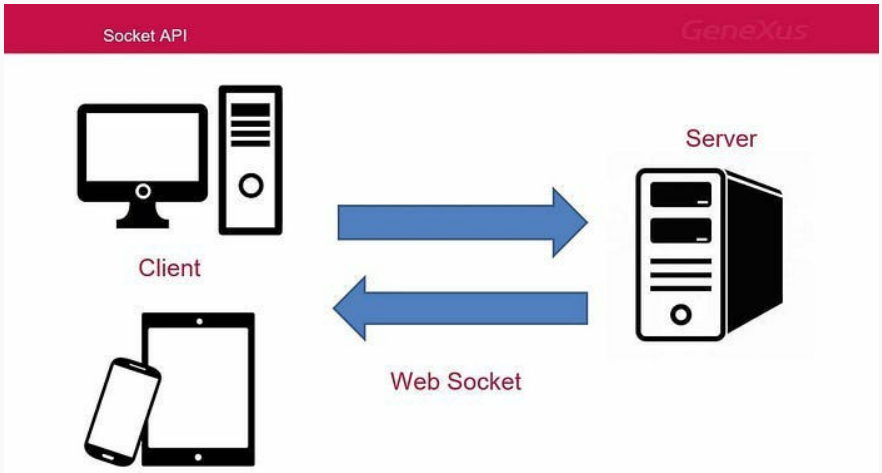
Daniel CraciuninLevel Up Coding

Stop Using UUIDs in Your Database

How UUIDs can Destroy SQL Database Performance

4 min read · May 17, 2024

600 53



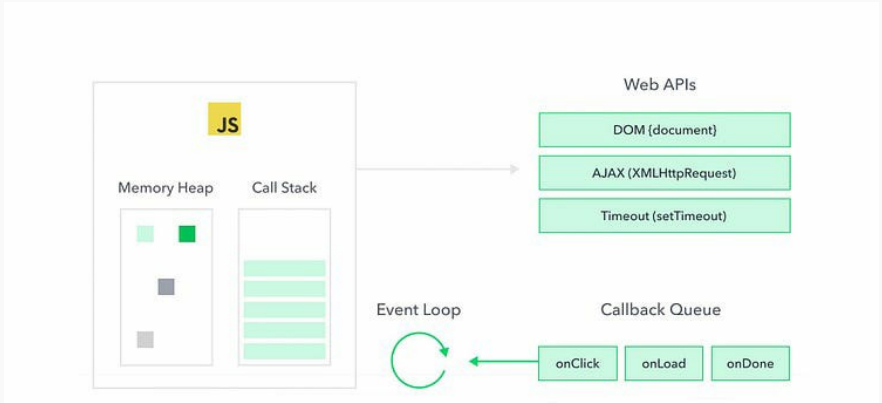
Rahul Anandeshi

Implementing Real-Time Notifications in a Node.js Express Application with PostgreSQL and Socket.io

Introduction:

3 min read · Mar 22, 2024

8



ShashankinBootcamp

Mastering the Event Loop: The Key to Asynchronous Programming

Event loop: The only article you need

4 min read · Feb 28, 2024

1

Minko Gechev in Angular Blog

Angular v18 is now available!

Today we are excited to share the next milestone in the evolution of Angular! Over the past three releases we've introduced a lot of new...

13 min read · 6 days ago

3.6K 26

See more recommendations