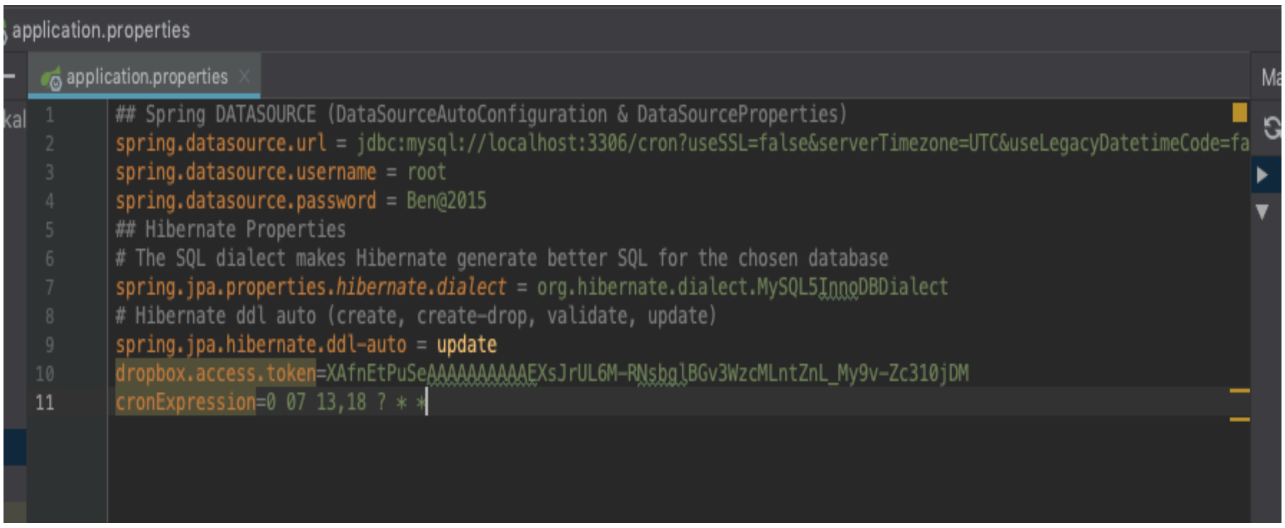
SCENARIO 1:

The Billing application runs at 07:00 once a month and export all successful and completed transactions to Drop box.

REQUIREMNT:
1.Please have MYQL DATABASE running on your machine.
2.In Application.properties kindly set MYSQL credentials
3. create a database called cron



```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url = jdbc:mysql://localhost:3306/cron?useSSL=false&serverTimezone=UTC&useLegacyDatetimeCode=fa
spring.datasource.username = root
spring.datasource.password = Ben@2015
## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
dropbox.access.token=XAfnEtPuSeAAAAAAAAAAEXsJrUL6M-RNsbglBGv3WzcMLntZnL_My9v-Zc310jDM
cronExpression=0 07 13,18 ? * *
```

JPA ORM, is used for storing, accessing, and managing Java objects in a relational database

Entity:

```java
import javax.persistence.*;
import java.time.LocalDateTime;

@Entity
public class Bill {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)

}
```

Repository:

```java
package co.za.absa.assesement.benjaminkalombo.repository;

import co.za.absa.assesement.benjaminkalombo.model.Bill;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface BillRepository extends JpaRepository<Bill, Long> {
}
```

Service:

```java
package co.za.absa.assesement.benjaminkalombo.service;

import co.za.absa.assesement.benjaminkalombo.model.Bill;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
```

```java
public interface BillService {
   void addBill(Bill bill);
```

TASK SCHEDULER That runs at specific time which read from database then export the report to drop box. (ONCE A MONTH AT 07:00 AM)

```java
@Component
public class CronTaskScheduler implements CommandLineRunner {

    private final FileExporter fileExporter;
    private final DropBox dropBox;
    private final BillService billService;
    public CronTaskScheduler(FileExporter fileExporter, DropBox dropBox,
BillService billService){
        this.fileExporter = fileExporter;
        this.dropBox = dropBox;
        this.billService = billService;
    }
    @Scheduled(cron = "0 7 1 * *”)
    public void process(){
        try {
            uploadFile();
        } catch (IOException | DbxException e) {
            e.printStackTrace();
        }
    }
```

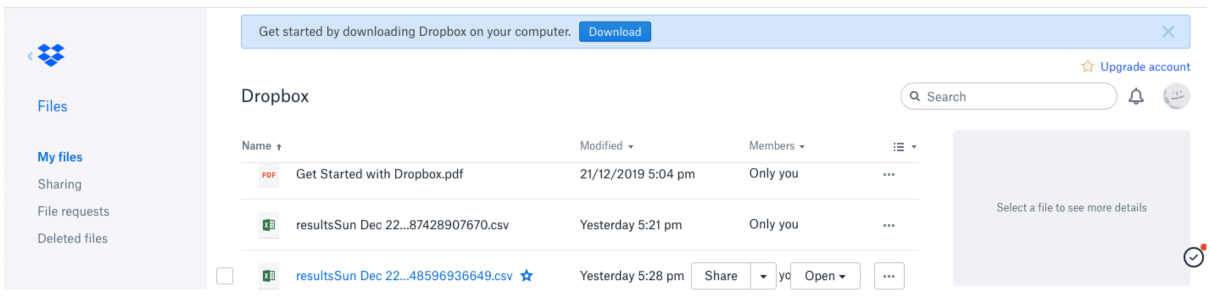
Export file to csv:

```java
private File exportCSV(final List<Bill> billList) throws IOException {
    File tempFile = null;
    if (billList.isEmpty()) {
        System.out.println("File is empty");
    } else {

        final String CSV_SEPARATOR = ",";
        final String tmpdir = System.getProperty("java.io.tmpdir");
        tempFile = File.createTempFile(tmpdir + "/results" + new Date().toString()
+ "BillingReport", ".csv");
        try {
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(tempFile), StandardCharsets.UTF_8));
            for (Bill bill : billList) {
                // String status = promotion.getStatus();

                    StringBuffer oneLine = new StringBuffer();
                    oneLine.append(bill.getClientswiftaddress().trim().length() ==
0 ? "" : bill.getClientswiftaddress());
                    oneLine.append(CSV_SEPARATOR);
                    oneLine.append(bill.getTransactionreference().trim().length()
== 0 ? "" : bill.getTransactionreference());
                    oneLine.append(CSV_SEPARATOR);
                    oneLine.append(bill.getCurrency().trim().length() == 0 ? "" :
bill.getCurrency());
                    oneLine.append(CSV_SEPARATOR);
                    oneLine.append(bill.getAmount() == 0 ? "" : bill.getAmount());
                    oneLine.append(CSV_SEPARATOR);
                    oneLine.append(bill.getMessagestatus().trim().length() == 0 ?
"" : bill.getMessagestatus());
                    oneLine.append(CSV_SEPARATOR);
                    oneLine.append(bill.getDatetimecreated() == null ? "" :
bill.getDatetimecreated());
                    oneLine.append(CSV_SEPARATOR);
                    bw.write(oneLine.toString());

                    bw.newLine();
            }
            bw.flush();
            bw.close();
        } catch (UnsupportedEncodingException | FileNotFoundException e) {

        } catch (IOException e) {

        }
    }
    return tempFile;
}
```

DROPBOX bean
Which upload file to Drop box

```java
private void init(){
   config = new DbxRequestConfig("absa/api-upload", "en_US");
    ACCESS_TOKEN = "XAfnEtPuSeAAAAAAAAAAEXsJrUL6M-RNsbglBGv3WzcMLntZnL_My9v-
Zc310jDM";
   client = new DbxClientV2(config, ACCESS_TOKEN);
}

public void uploadFile(File file) throws IOException, DbxException {
    System.out.println("uploadignfile to dropbox..");
    try (InputStream in = new FileInputStream(file)) {
        FileMetadata metadata = client.files().uploadBuilder("/" + file.getName() )
                .uploadAndFinish(in);
        System.out.println("Done uploading file "+ file.getName() + " to DropBox");
    }
    file.deleteOnExit();
}
```

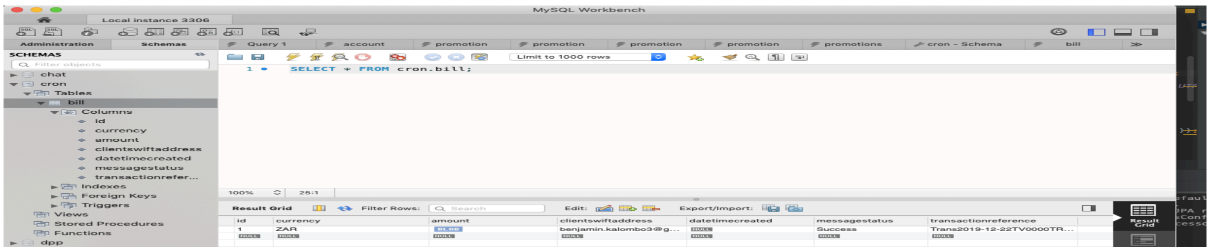
BILLING REPORT exported to DROP BOX:


Get started by downloading Dropbox on your computer.
Download
Files
My files
Sharing
File requests
Deleted files
Upgrade account
Search
Select a file to see more details
Share
Open

Dropbox

Name	Modified	Members
Get Started with Dropbox.pdf	21/12/2019 5:04 pm	Only you
resultsSun Dec 22...87428907670.csv	Yesterday 5:21 pm	Only you
resultsSun Dec 22...48596936649.csv	Yesterday 5:28 pm	

DB:



```sql
SELECT * FROM cron.bill;
```

id	currency	amount	clientswiftaddress	datetimecreated	messagestatus	transactionreference
1	ZAR	BLOB	benjamin.kalombo3@g...	NULL	Success	Trans2019-12-22TV0000TR...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

SCENARIO 2:
What is a stream: these are sequences of objects represented as a conduit of data., the stream never modify the underlying data rather operates on a source such as a an Array or collection
a-Sequential Stream

The stream is processed sequentially, they do not use a multicore system even when we use a multithreading to process the stream? It will operate on a single core at a time.

i.e
List<String> statuses = Arrays.asList("Received", "Pending", "","Awaiting Maturity", "Completed","");

```
public static void main(String[] args) throws DbxException, IOException {

            List<String> statuses = Arrays.asList("Received", "Pending",
"","Awaiting Maturity", "Completed","");
            statuses.stream().forEach(System.out::println);
            System.out.println();
          //  list.parallelStream().forEach(System.out::println);


    SpringApplication.run(BenjaminkalomboApplication.class, args);
}
```

This example is an illustration of sequential stream. The *list.stream()* works in sequence on a single thread with the *println()* operation.
output:

```
Received
Pending

Awaiting Maturity
Completed
```

b-Parallel Stream:
Parallel stream leverage multicore processors, resulting in a substantial increase in performance.
To ensure that the result of parallel processing applied on stream is same as is obtained through sequential processing, parallel streams must be stateless, non-interfering, and associative.

```
public static void main(String[] args) throws DbxException, IOException {

            List<String> statuses = Arrays.asList("Received", "Pending",
"","Awaiting Maturity", "Completed","");
    statuses.parallelStream().forEach(System.out::println);


    SpringApplication.run(BenjaminkalomboApplication.class, args);
}
```

output:

```
Pending

Completed
Awaiting Maturity
Received
```

c-Filter : remove all non -empty string from list

```
public static void main(String[] args) throws DbxException, IOException {

            List<String> statuses = Arrays.asList("Received", "Pending",
"","Awaiting Maturity", "Completed","");
    statuses = statuses.stream().filter(item->
!item.isEmpty()).collect(Collectors.toList());
```

```java
        statuses.parallelStream().forEach(System.out::println);
```

output:

```
Pending
Received
Completed
Awaiting Maturity
```

=================================== END ===================================