Online Video Streaming Platform

Long term Internship Project Submitted
in partial fulfillment of the requirements for the award of the degree
of

BACHELOR OF TECHNOLOGY

Submitted By

A N Jagadeeswaran
ID NO : R170661

Under the supervision of

Ms. C Suneetha
(Assistant Professor)

Department of Computer Science Engineering

Rajiv Gandhi University Of Knowledge Technologies(RGUKT)
R.K Valley , Kadapa , Andhra Pradesh

Rajiv Gandhi University Of KnowledgeTechnologies
R.K Valley , Kadapa , Andhra Pradesh

CERTIFICATE

This is to certify that report entitled **“Online Video Streaming Platform”** submitted by **A N Jagadeeswaran**(R170661) in partial fulfilment of the requirements for the degree of award of bachelor of technology in computer science engineering is a bonafide work carried by him under my supervision and guidance.

The report has been not submitted previously in part or full to this university or any other university or institution for the award of any degree or diploma.

INTERNAL GUIDE	**HEAD OF THE DEPARTMENT**
Ms. C Suneetha	Mr. N Satyanandram
Assistant Professor	HOD OF CSE

DECLARATION

I , hereby declare that this report entitled **" Online Video Streaming Platform "** submitted by me under the guidance and supervision of **Ms C Suneetha** , is a bonafide work . I also declare that it has not been submitted previously in part or in full to this university or any other university or institution for the award of any degree or diploma.

Date : - (A N Jagadeeswaran)

Place : - RK Valley (R170661)

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and who's constant guidance and encouragement crown all the efforts success.

I would like to express my sincere gratitude to **Ms. C Suneetha ,** my project guide for valuable suggestions and keen interest throughout the progress of the project.

I'm grateful to **Mr. N. Satyanandram , HOD CSE** , for providing excellent computing facilities and congenial atmosphere for progressing the project.

At the outset, I would like to thank **Honourable Director Madam , Ms. K Sandhya Rani** , for providing all the necessary resources and support for the successful completion of my course work.

TABLE OF CONTENTS

PROBLEM STATEMENT

1) Create CRUD operations for the table tblContentAds of the production code

2) Create CRUD operations for the table tblPartnerContentDeeplinks of the production code

3) Create CRUD operations for the table tbleTokens used for token generation in the production code

4) Create operations for getting SonyLiv content

ABSTRACT

YuppTV is an over-the-top (OTT) content provider for South Asian content including live television and films with recording and storage features. YuppTV allows broadcasters and content providers to reach an audience, and allows consumers to view content on up to six screens of connected TVs, STBs, PC, smartphones, tablets and game console.

YuppTV is a revolutionary and pioneer in providing a no-holds- bar gateway for television viewers from across the globe. With the power of technology and convenience of the internet, YuppTV has allowed consumers to view the latest Television Content Live anytime and anywhere.

The boundaries of the television viewing experience have been pushed so far that anyone can gain access to Live TV Channels, Catch-Up TV, and Unlimited Movies at their convenience. YuppTV has made the availability of Indian TV Channels easier and cost effective all over the world.

As a software developer Intern I'm working on backend part of the YuppTV Website and Projects where Scala ,REST API's, SLICK , POSTGRESQL and some tools are used for backend support.

INTRODUCTION

1) CRUD Operations for tblContentAds

The tblContentAds is a relational database table that stores the information related to Ads. There are two types of Ads when coming to YuppTv Services. They are :

1) **Pre-roll ad :-** These are the ads that are displayed before a content starts playing

2) **Mid-roll ad :-** These are the ads that are displayed in between the playing of the media file.

CRUD operations includes create , read , update and delete operations.

In the production code , in order to manage the contentAds there is a necessity to write CRUD operations using **SLICK.** As part of production code we need to put certain restrictions on the code like Users who belongs to India should not get any Ads while using the YuppTv application and this is not applicable for any other country users and this is doen based on the location information accessed from the user.

2) CRUD Operations for tblPartnerContentDeeplinks

The tblPartnerContentDeeplinks is a relational database table that stores the content such as movies , web series , new related to the partnered services . In order to provide them with functionalities of reading the data , updating the data and deleting the data related to their content these CRUD operations have been written using SLICK .

Operations done by Content providers : -

1) Read

2) Write/Update

3) Delete

3) CRUD operations for tblTokens

The table tblTokens ia relational database table that stores information related to user sessions. A session begins when a user gets access to the website that is a session begins when a token is generated.

tblTokens stores information such as device_id , device_type , login_id and session_id. A user can access the website only after the successful generation of the token. If the token is not generated then users will be denied access.

4) Operations for getting sonLiv content

SonyLiv provides or shares content with the YuppTv to stream on its platforms. To produce those content to the users we need to extract them individually like webseries should be separated from the content of the movies and from the content of the news channels and also it includes cricket live channels.

In order to extract them certain operations has to be performed so that dat is individually stored in the individual tables based on their category like webseries , live tv , movies , tv shows etc.

TOOLS USED

SCALA :

Scala combines object-oriented and functional programming in one concise, high-level language. Scala's static types help avoid bugs in complex applications, and its JVM and JavaScript runtimes let you build high-performance systems with easy access to huge ecosystems of libraries.

REST API's :

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer .

SLICK :

Slick is a Functional Relational Mapping library for Scala that allows us to query and access a database like other Scala collections. We can write database queries in Scala instead of SQL, thus providing typesafe queries.

POSTGRESQL :

PostgreSQL comes with many features aimed to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help you manage your data no matter how big or small the dataset. In addition to being free and open source, PostgreSQL is highly extensible. For example, you can define your own data types, build out custom functions, even write code from different programming languages without recompiling your database.

REQUIREMENTS SPECIFICATION

Name	: Online Video Streaming Platform
Model	: Y001
CPU	: ARM Cortex A9 (Dual-Core 1.5 GHZ)
Operation Sytem	: Android 4.2
Memory	: 1 GB
Flash Memory	: 4 GB
Enterprise Standard	: Q/SCWR 021-2013
Power Supply	: SV = 2A
Working Environment	: Working

SOURCE CODE

ContentAdsHandler

```
class ContentAdsHandler extends BaseRequestHandler {


  private val Log = LoggerFactory getLogger getClass
  supportedOperations = (Set(Operation.List, Operation.Get, Operation.Insert,
Operation.Update, Operation.Delete, Operation.Fetch))
  handlerPermission = Some(Permissions.ContentAds)


  override def executeListRequest(req: ListRequest, user: UserDetails) = {
    Log info ("Channel Group Handler -- executeList called")
    try {
      val contentAdsFuture =
DBHelper.getAllContentAds(req.page.pageNumber, req.page.pageSize)
      contentAdsFuture.map { x => CommonEntityList(x) }
    } catch {
      case t: Throwable => Future { CommonEntityList(List()) }
    }
  }


  override def executeInsert(req: InsertRequest, user: UserDetails) = {
    Log info ("contentAds -> executeInsert called")
    val data = req.dataJson
    try {
      val contentAdsData = data.asJson.convertTo[ClientContentAds]
```

```
          val insertContentAdsFuture =
DBHelper.insertContentAds(contentAdsData, user)
          insertContentAdsFuture.map(InsertResponse(_))
    } catch {
      case t: DeserializationException => {
        Log info ("Error Message: " + t.getMessage + ": \n stackTrace :" +
t.getStackTrace.toString())
        Future(Error(ErrorCodes.ClientDataError,
ErrorMessages.ClientDataError))
      }
      case t: Throwable => {
        Log info ("Error Message: " + t.getMessage + ": \n stack trace :" +
t.getStackTrace)
        Future(Error(ErrorCodes.UnknownError,
ErrorMessages.UnknownError))
      }
    }
  }

  override def executeUpdate(req: UpdateRequest, user: UserDetails) = {
    Log info ("Content Ads handler -- executeUpdate called")
    val data = req.dataJson
    try {
      val contentAdsData = data.asJson.convertTo[ClientContentAds]
          val insertSportsSeriesFuture =
DBHelper.updateContentAds(contentAdsData, user)
          insertSportsSeriesFuture.map(InsertResponse(_))
    }
```

```
catch {
      case t: DeserializationException => {
        Log info ("Error Message: " + t.getMessage + ": \n stackTrace :" +
t.getStackTrace.toString())
        Future(Error(ErrorCodes.ClientDataError,
ErrorMessages.ClientDataError))
      }
      case t: Throwable => {
        Log info ("Error Message: " + t.getMessage + ": \n stack trace :" +
t.getStackTrace)
        Future(Error(ErrorCodes.UnknownError,
ErrorMessages.UnknownError))
      }
    }
  }

  override def executeGetRequest(req: GetRequest, user: UserDetails) = {
    Log info ("Content Ads Handler-- executeGet called")
    try {
      val contentAdsDetails = DBHelper.getContentAdsDetails(req.entityId)
      contentAdsDetails.map { x => CommonEntityList(x) }
    } catch {

      case t: Throwable => Future { CommonEntityList(List()) }
    }
  }
```

```
override def executeDelete(req: DeleteRequest, user: UserDetails) = {
    Log info ("content Ads Handler -- executeDelete called")
    val cid = req.id
    val deleteContentAdsFuture = DBHelper.deleteContentAds(cid)
    deleteContentAdsFuture.map(DeleteResponse(_))
  }

  override def executeFetchRequest(req: FetchRequest, user: UserDetails) = {
    Log info ("Package Tabs Handler  -> executeFetch called")
    val clientFilter = req.dataJson
    Log info ("JSON data: +" + clientFilter)
    val filter = clientFilter.asJson.convertTo[UserFilter]
    val allContentAds = DBHelper.getAllContentAds()
    val fetchContentAdsFuture = applyUserFilter(allContentAds, filter)
    fetchContentAdsFuture.map(CommonEntityList(_))
  }

    }
  }
```

PartnerContentDeeplinks

```
class PartnerContentDeeplinksHandler extends BaseRequestHandler {


  private val Log = LoggerFactory getLogger getClass
  supportedOperations = (Set(Operation.List, Operation.Get, Operation.Insert,
Operation.Update, Operation.Delete, Operation.Fetch))
  handlerPermission = Some(Permissions.ChannelGroup)


  override def executeListRequest(req: ListRequest, user: UserDetails) = {
    Log info ("Partner Content Deeplinks Handler -- executeList called")
    try {
      val partnerContentDeeplinksFuture =
DBHelper.getAllPartnerContentDeeplinks(req.page.pageNumber,
req.page.pageSize)
      partnerContentDeeplinksFuture.map { x => CommonEntityList(x) }
    } catch {
      case t: Throwable => Future { CommonEntityList(List()) }
    }
  }
  override def executeInsert(req: InsertRequest, user: UserDetails) = {
    Log info ("partnerContentDeeplinks -> executeInsert called")
    val data = req.dataJson
    try {
      val partnerContentDeeplinksData =
data.asJson.convertTo[ClientPartnerContentDeeplinks]
```

```
      val codeExistsFuture =
DBHelper.doesContentExists(partnerContentDeeplinksData.contentId,
partnerContentDeeplinksData.partnerId,
partnerContentDeeplinksData.deviceId)
      val codeExists = Await.result(codeExistsFuture, Duration.Inf)
      codeExists match {
        case true => {
          Log error ("Error Message: " + (ErrorCodes.DuplicateRecord,
ErrorMessages.DuplicateRecord))
          Future(Error(ErrorCodes.DuplicateRecord,
ErrorMessages.DuplicateRecord))
        }
        case false => {
          val insertPartnerContentDeeplinksFuture =
DBHelper.insertPartnerContentDeeplinks(partnerContentDeeplinksData, user)
          insertPartnerContentDeeplinksFuture.map(InsertResponse(_))
        }
      }
    }
      }
    }
  override def executeUpdate(req: UpdateRequest, user: UserDetails) = {
    Log info ("Partner Content Deeplinks -- executeUpdate called")
    val data = req.dataJson
    try {
      val partnerContentDeeplinksData =
data.asJson.convertTo[ClientPartnerContentDeeplinks]
```

```
      val codeExistsFuture =
DBHelper.doesContentExists(partnerContentDeeplinksData.contentId,
partnerContentDeeplinksData.partnerId,
partnerContentDeeplinksData.deviceId)
      val codeExists = Await.result(codeExistsFuture, Duration.Inf)
      codeExists match {
        case true => {
          Log error ("Error Message: " + (ErrorCodes.DuplicateRecord,
ErrorMessages.DuplicateRecord))
          Future(Error(ErrorCodes.DuplicateRecord,
ErrorMessages.DuplicateRecord))
        }case false => {
          val insertContentDeeplinksFuture =
DBHelper.updatePartnerContentDeeplinks(partnerContentDeeplinksData, user)
          insertContentDeeplinksFuture.map(InsertResponse(_))
        }
      }
    }
      }
    }
  }
  override def executeDelete(req: DeleteRequest, user: UserDetails) = {
    Log info ("partner Content Deeplinks Handler -- executeDelete called")
    val cid = req.id
    val deletePartnerContentDeeplinksFuture =
DBHelper.deletePartnerContentDeeplinks(cid)
    deletePartnerContentDeeplinksFuture.map(DeleteResponse(_))
  }
```

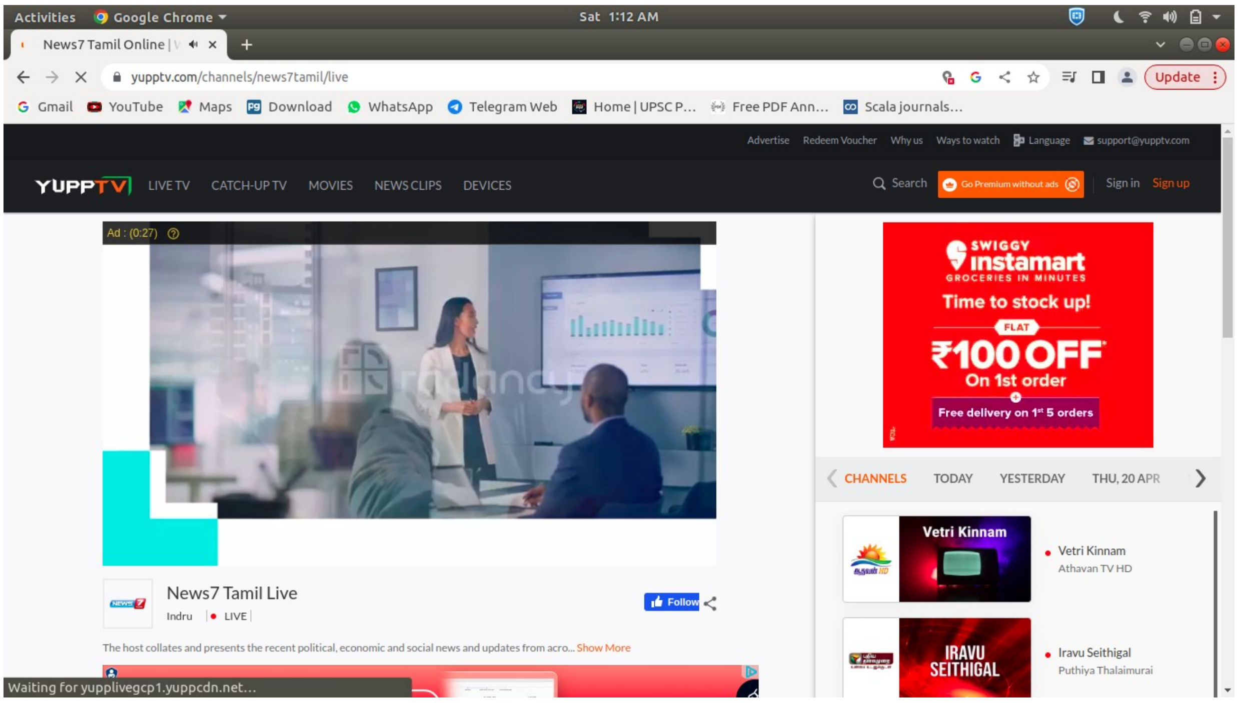
SonyLivContent

```
private def publish(content: List[PartnerContentArchive], contentType: String,
jobName: String, pageSize: Int, pageNum: Int = 0): Unit = {
    implicit val ec = Bootstrapper.getPartnerBgTaskDispatcher
    Log info ("total " + contentType + " size => " + content.size)
    val data = content.drop(pageNum * pageSize).take(pageSize)
    Log info ("publish " + contentType + " size =>" + data.size)

    contentType match {
      case "movies" => {
        Log info ("start - publish movies size : " + data.size + " , page : "
+ pageNum)
        val startTime = System.currentTimeMillis
        SonyLivServices.integrateMovies(data).onComplete {
          case Success(res) => {
            Log info ("Successfully Processed Movies : page : " + pageNum
+ " and time taken: " + (System.currentTimeMillis - startTime) / 1000.0)

            if (data.size > 0 && data.size == pageSize) {
              publish(content, contentType, jobName, pageSize, pageNum +
1)
            }
          }
          case Failure(err) => Log info ("Error publishing movies : " +
data.map(_.displayFields.name) + " ,err :" + err)
        }
        Log info ("complete - published movies size : " + data.size + " ,
page : " + pageNum)
        Thread.sleep(3000)
        Log info ("next 1")
      }
      case "shows" => {
        Log info ("start - publish shows size : " + data.size + " , page : " +
pageNum)
        val startTime = System.currentTimeMillis
        SonyLivServices.integrateShows(data).onComplete {
          case Success(res) => {
```

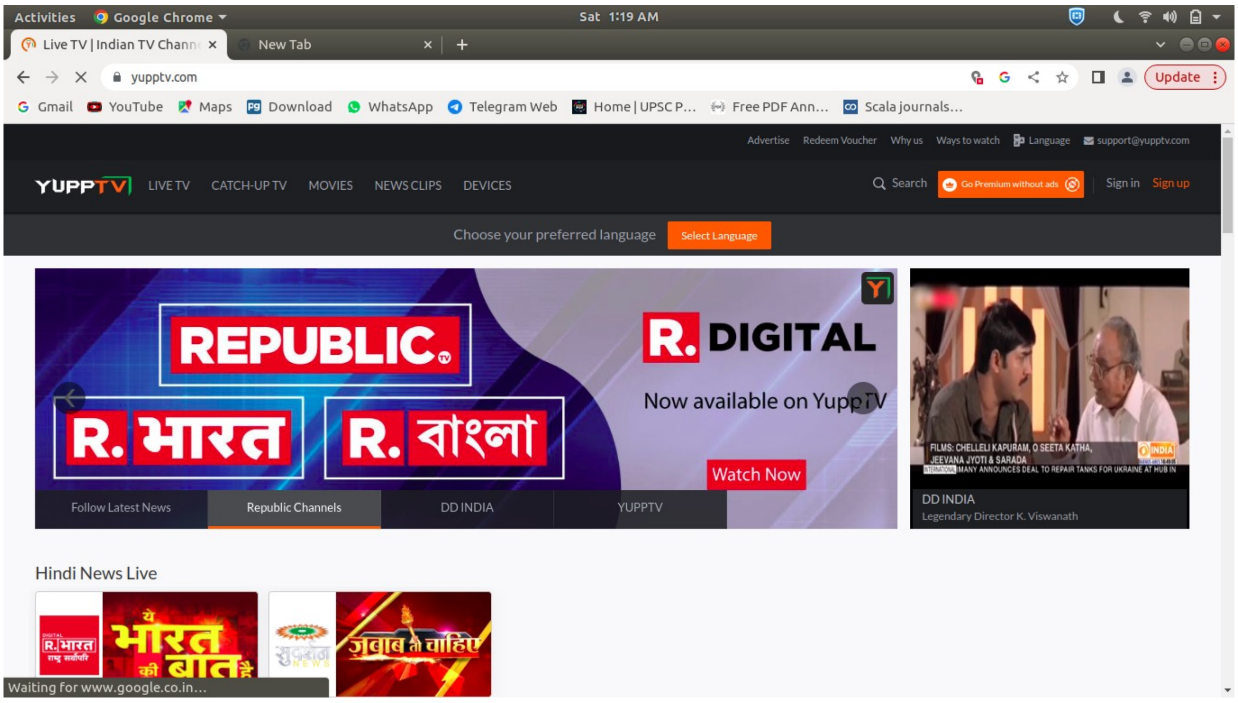
```
Log info ("Successfully Processed Shows : page :" + pageNum + " and time
taken: " + (System.currentTimeMillis - startTime) / 1000.0)
            if (data.size > 0 && data.size == pageSize) {
              publish(content, contentType, jobName, pageSize, pageNum +
1)
            }
          }
          case Failure(err) => Log info ("Error publishing shows : " +
data.map(_.displayFields.name) + " ,err :" + err)
        }
        Log info ("complete - published shows size : " + data.size + " ,
page : " + pageNum)
        Thread.sleep(3000)
        Log info ("next 2")
      }
      case "episodes" => {
        Log info ("start - publish episodes : size : " + data.size + " , page :
" + pageNum)
        val startTime = System.currentTimeMillis
        SonyLivServices.integrateEpisodes(data).onComplete {
          case Success(res) => {
            Log info ("Successfully Processed Episodes : page : " + pageNum
+ " and time taken: " + (System.currentTimeMillis - startTime) / 1000.0)
            if (data.size > 0 && data.size == pageSize) {
              publish(content, contentType, jobName, pageSize, pageNum +
1)
            } else GenericSchema.updateJobDetails(jobName)
          }

SonylivSportsServices.integrateTournamentBundleData(data).onComplete {

case Success(res) => {
            Log info ("Succesfully Processed Tournamentvideos : page : " +
pageNum)


      GenericSchema.updateJobDetails(jobName)
          }
          case Failure(err) => Log info ("Error publishing tournamentvideos
: " + data.map(_.displayFields.name) + " ,err :" + err)
        }
        Log info ("complete - published tournamentvideos size : " +
data.size + " , page : " + pageNum)
        Thread.sleep(200)
        Log info ("next 4")
      }
      case _ => Log info ("No matching content type found.")
    }
  }
  private def toPartnerContentArchive(p: ClientPartnerContentArchive,
partnerContent: Option[(Option[Long], String, Timestamp)]) = {
    val (yuppContentId, createdBy, createdDate) = if
(partnerContent.isDefined) partnerContent.get else (None, "System", new
java.sql.Timestamp(System.currentTimeMillis()))
    val (display, links, images) = (p.displayFields, p.deepLinks, p.imageUrls)
    val deepLinks = TMDeepLinks(links.webdeeplinkUrl, links.deeplinkUrl,
links.playerDeeplinkUrl)

  }
}
```

RESULT

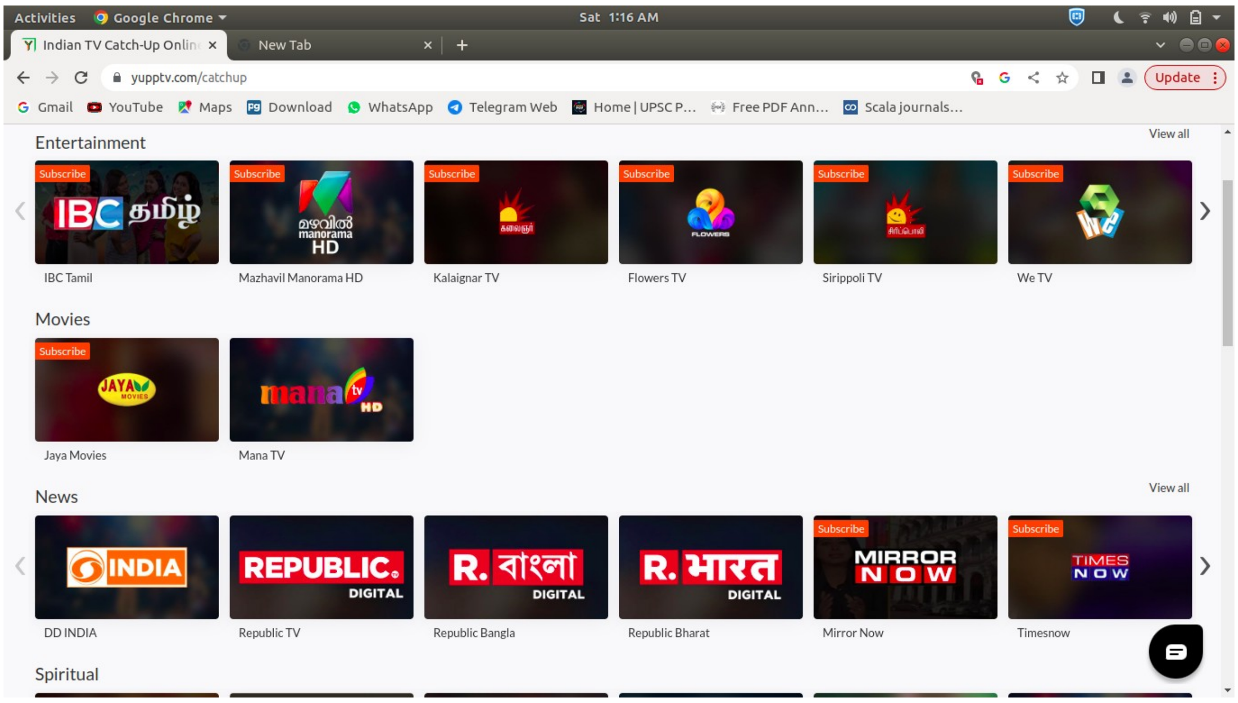
Content Ads

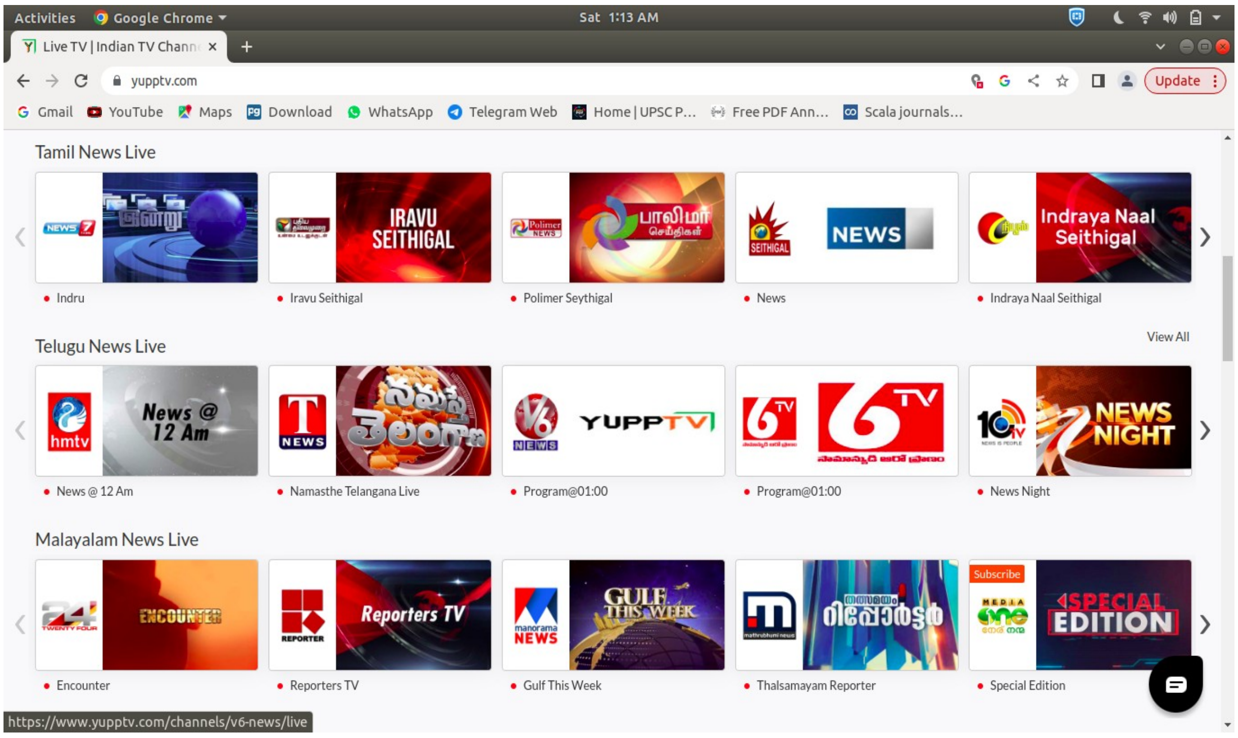


The host collates and presents the recent political, economic and social news and updates from acro... Show More



Content management from Providers





CONCLUSION

From a proper analysis of positive points and constraints on the Online Video Streaming Platform (OVSP) , it can be safely concluded that the project OVSP is a highly e-client GUI based component.

In modern times , it is hard to find an application that integrates all the contents from different sources i.e., different content providers . But this online video streaming platform has been mainly developed keeping in mind that users should be able to access the application with ease by allowing broadcasters and content providers to reach an audience , and allows consumers to view content on upto six screens.

REFERENCES

1. https://www.scala-lang.org/
2. https://stackoverflow.com/questions/tagged/scala
3. https://scala-slick.org/
4. https://akka.io/
5. https://www.udemy.com/course/akka-essentials/
6. https://www.udemy.com/course/akka-http/