

SPRINT 2: Desarrollo del Backend

Identificación Proyecto

Nombre Proyecto:	EstoEsCine
Número Equipo:	01
Integrantes del equipo	
Rol (Líder-Desarrollador – Cliente)	Nombre
Líder/desarrollador	Daniel Fernando Gil García
Desarrollador	Paula Andrea Gutiérrez Avendaño
Desarrollador/cliente	Melissa Girón Rodríguez

Evidencia construcción del Backend

Captura de código CONTROLLERS MODEL

GET

Actores:

```

// actoresCtrl.js
const actoresModel = require('../models/actoresSchema');

const actoresListar = async (req, res) => {
  let actores = await actoresModel.find();
  res.send(actores);
}

module.exports = {
  actoresListar
}

// actoresSchema.js
const mongoose = require('mongoose');

const actoresSchema = mongoose.Schema(
  {
    nombre: {
      type: String,
      require: true,
      trim: true
    }
  }
);

module.exports = mongoose.model('actores', actoresSchema);

```



```
[{"_id": "6371722a58ee4060a2bb60f4", "nombre": "Arnold Schwarzeneger"},
{"_id": "6371724f58ee4060a2bb60f5", "nombre": "Linda Hamilton"},
{"_id": "6371727558ee4060a2bb60f6", "nombre": "Leonardo Di Caprio"},
{"_id": "6371728958ee4060a2bb60f7", "nombre": "Kate Winslet"},
{"_id": "637287279eb3dbaa58747feb", "nombre": "Brendan Fraser"},
{"_id": "637287439eb3dbaa58747fec", "nombre": "Rachel Weisz"},
{"_id": "637288ab9eb3dbaa58747ffb", "nombre": "Miyu Irino"},
{"_id": "637288b29eb3dbaa58747ffc", "nombre": "Rumi Hiiragi"}]
```

Ítems:

JS itemsCtrl.js

controllers > JS itemsCtrl.js > ...

```

1  const itemsModel = require ('../models/itemsSchema')
2
3
4  const itemsListar = async (req,res) => {
5      let items = await itemsModel.find()
6      res.send(items);
7  }
8  module.exports = {
9      itemsListar
10 }

```

JS itemsSchema.js

models > JS itemsSchema.js > ...

```

1  const mongoose = require ('mongoose')
2
3  const itemsSchema = mongoose.Schema(
4      {
5          nombre:{
6              type : String,
7              require : true,
8              trim : true
9          }
10     }
11 )
12
13 module.exports = mongoose.model("items", itemsSchema)

```

DLlo Software MINTIC

el viaje de chihiro - Go

Rumi Hiiragi - Wikiped

localhost:3000/api/ite

localhost:3000/api/items

```

[{"_id":"6370ef19e6620cc673652b1e","titulo":"Titanic","tipo":"Pelicula","genero":"Romance","año":"1997","duracion":195,"created_at":"2022-11-12","creado_por":"admin","actores":[{"nombre":"Leonardo Di Caprio"}, {"nombre":"Kate Winslet"}]}, {"_id":"6371715358ee4060a2bb60ec","titulo":"Terminator","tipo":"Pelicula","genero":"Ciencia Ficción","año":"1984","duracion":108,"created_at":"2022-11-12","creado_por":"admin","actores":[{"nombre":"Arnold Schwarzeneger"}, {"nombre":"Linda Hamilton"}]}, {"_id":"637286f59eb3dbaa58747fe5","titulo":"La Momia","tipo":"Pelicula","genero":"Aventura","año":"1999","duracion":124,"created_at":"2022-11-14","creado_por":"admin","actores":[{"nombre":"Brendan Fraser"}, {"nombre":"Rachel Weisz"}]}, {"_id":"637288859eb3dbaa58747ff5","titulo":"El Viaje de Chihiro","tipo":"Pelicula","genero":"Fantasía","año":"2001","duracion":125,"created_at":"2022-11-14","creado_por":"admin","actores":[{"nombre":"Rumi Hiiragi"}, {"nombre":"Miyu Irino"}]}]

```

Géneros:

JS generosCtrl.js

controllers > JS generosCtrl.js > [?] <unknown>

```

1  const generoModel = require ('../models/generosSchema')
2
3
4  const generoListar = async (req,res) => {
5      let generos = await generoModel.find()
6      res.send(generos);
7  }
8  module.exports = {
9      generoListar
10 }

```

JS generosSchema.js

models > JS generosSchema.js > ...

```

1  const mongoose = require ('mongoose')
2
3  const generoSchema = mongoose.Schema(
4      {
5          nombre:{
6              type : String,
7              require : true,
8              trim : true
9          }
10     }
11 )
12
13 module.exports = mongoose.model("genero", generoSchema)

```



```
[{"_id":"637171b758ee4060a2bb60f0","nombre":"Romance"},
{"_id":"637171ea58ee4060a2bb60f1","nombre":"Ciencia Ficción"},
{"_id":"637287189eb3dbaa58747fe8","nombre":"Aventura"},
{"_id":"637288939eb3dbaa58747ff8","nombre":"Fantasía"}]
```

POST

items:

```
8 //Guardar ITEMS
9 const itemsGuarda = (req,res) =>{
10   console.log(req.body)
11   try {
12     const item = new itemsModel (req.body)
13     item.save()
14     res.send("item guardado")
15   }
16   catch (err) {
17     console.log("error itemsGuarda: " + err)
18   }
19 }
```

Actores:

```
9 //post
10 const actoresGuarda = (req,res) =>{
11   console.log(req.body)
12   try {
13     const actores = new actoresModel (req.body)
14     actores.save()
15     res.send("Actores guardados")
16   }
17   catch (err) {
18     console.log("error actoresGuarda: " + err)
19   }
20   //res.send("ok")
21 }
22 }
```

Géneros:

```
//post
const generoGuarda = (req,res) =>{
  console.log(req.body)
  try {
    const genero = new generoModel (req.body)
    genero.save()
    res.send("Genero guardado")
  }
  catch (err) {
    console.log("error GeneroGuarda: " + err)
  }
  //res.send("ok")
}
```

PUT

Items:

```
controllers > JS itemsCtrl.js > [0] itemsActualizar
24 const itemsActualizar = async (req,res) =>{
25   console.log(req.body)
26   try {
27     const {id, titulo, duracion, genero, año, actores} = req.body
28
29     if (id == ''){
30       res.status(400).send("id de Item No Valido")
31     }
32
33     if (titulo == '') {
34       res.status(400).send("Titulo de item No Valido")
35     }else {
36       //res.send("ok")
37       const item = {}
38       item.titulo = titulo
39       item.duracion = duracion
40       item.genero = genero
41       item.año = año
42       item.actores = actores
43
44       const rta = await itemsModel.updateOne(
45         { _id : id},
46         { $set : item},
47         { new : true}
48       )
49       res.status(200).json({"msj": "Item Actualizado"})
50
51     }
52   }
53   catch (err) {
54     console.log("Error itemsActualizar: " + err)
55   }
56 }
```

Actores:

```

25 // put
26 const actoresActualizar = async (req,res) =>{
27   console.log(req.body)
28   try {
29     const {id, nombre} = req.body
30
31     if (id == ''){
32       res.status(400).send("id de actor No Valido")
33     }
34
35     if (nombre == '') {
36       res.status(400).send("Nombre de actor No Valido")
37     }else {
38       //res.send("ok")
39       const actor = {}
40       actor.nombre=nombre
41
42       const rta = await actoresModel.updateOne(
43         { _id : id},
44         { $set : actor},
45         { new : true}
46       )
47       res.status(200).json({"msj": "actor Actualizado"})
48     }
49   }
50 }
51 catch (err) {
52   console.log("Error actoresActualizar: " + err)
53 }
54 }

```

Géneros:

```

25 const generoActualizar = async (req,res) =>{
26   console.log(req.body)
27   try {
28     const {id, nombre} = req.body
29
30     if (id == ''){
31       res.status(400).send("id de genero No Valido")
32     }
33
34     if (nombre == '') {
35       res.status(400).send("Titulo de genero No Valido")
36     }else {
37       //res.send("ok")
38       const genero = {}
39       genero.nombre = nombre
40
41       const rta = await generosModel.updateOne(
42         { _id : id},
43         { $set : genero},
44         { new : true}
45       )
46       res.status(200).json({"msj": "Genero Actualizado"})
47     }
48   }
49 }
50 catch (err) {
51   console.log("Error generosActualizar: " + err)
52 }
53 }

```

DELETE

items:

```
//delete
const itemsEliminar = async(req,res) =>{
  const id = req.params.id
  try {
    if (id == '') {
      res.status(400).send("el id No es valido")
    }else{
      const rta = await itemsModel.deleteOne({ _id : id})
      res.status(200).send("item Eliminado con exito")
      console.log("eliminado: " + id)
    }
  } catch (err) {
    console.log("error itemsEliminar: " + err)
  }
  //console.log(id)
  //res.send("eliminando...")
}
```

Actores:

```
56 //delete
57 const actoresEliminar = async(req,res) =>{
58   const id = req.params.id
59   try {
60     if (id == '') {
61       res.status(400).send("el id No es valido")
62     }else{
63       const rta = await actoresModel.deleteOne({ _id : id})
64       res.status(200).send("actor eliminado con exito")
65       console.log("eliminado: " + id)
66     }
67   } catch (err) {
68     console.log("error actoresEliminar: " + err)
69   }
70 }
71 }
```

Géneros:

```
55 //delete
56 const generoEliminar = async(req,res) =>{
57   const id = req.params.id
58   try {
59     if (id == '') {
60       res.status(400).send("el id No es valido")
61     }else{
62       const rta = await itemsModel.deleteOne({ _id : id})
63       res.status(200).send("Genero eliminado con exito")
64       console.log("eliminado: " + id)
65     }
66   } catch (err) {
67     console.log("error generosEliminar: " + err)
68   }
69 }
70 }
71 }
```

Evidencias de los “endpoint” con el consumo de recursos del API REST

Realizado por Thunder en Visual Code:

GET

Items:

The screenshot shows a REST client interface in Visual Studio Code. The request is a GET to `http://localhost:3000/api/items`. The response status is 200 OK, with a size of 1 KB and a time of 18 ms. The response body is a JSON array containing two movie objects.

Request:

- Method: GET
- URL: `http://localhost:3000/api/items`
- Body: (Empty)

Response:

```

1  [
2    {
3      "_id": "6370ef19e6620cc673652b1e",
4      "titulo": "Titanic",
5      "tipo": "Película",
6      "genero": "Romance",
7      "año": "1997",
8      "duracion": 195,
9      "created_at": "2022-11-12T00:00:00.000Z",
10     "creado_por": "admin",
11     "actores": [
12       {
13         "nombre": "Leonardo Di Caprio"
14       },
15       {
16         "nombre": "Kate Winslet"
17       }
18     ]
19   },
20   {
21     "_id": "6371715358ee4060a2bb60ec",
22     "titulo": "Terminator",
23     "tipo": "Película",
24     "genero": "Ciencia Ficción",
25     "año": "1984",
26     "duracion": 108,
27     "created_at": "2022-11-12T00:00:00.000Z",
28     "creado_por": "admin",
29     "actores": [

```



Actores:

JS generosCtrl.js TC localhost:3000/api/items X

GET localhost:3000/api/actores/ Send

Query Headers² Auth Body¹ Tests Pre Run^{New}

Query Parameters

☐ parameter value

Status: 200 OK Size: 489 Bytes Time: 58 ms

Response Headers⁶ Cookies Results Docs

```

11  "_id": "6371727558ee4060a2bb60f6",
12  "nombre": "Leonardo Di Caprio"
13  },
14  {
15    "_id": "6371728958ee4060a2bb60f7",
16    "nombre": "Kate Winslet"
17  },
18  {
19    "_id": "637287279eb3dbaa58747feb",
20    "nombre": "Brendan Fraser"
21  },
22  {
23    "_id": "637287439eb3dbaa58747fec",
24    "nombre": "Rachel Weisz"
25  },
26  {
27    "_id": "637288ab9eb3dbaa58747ffb",
28    "nombre": "Miyu Irino"
29  }

```

Géneros:

GET localhost:3000/api/generos Send

Query Headers² Auth Body Tests Pre Run^{New}

Json Xml Text Form Form-encode GraphQL Binary

1

Status: 200 OK Size: 176 Bytes Time: 13 ms

Response

```

6  {
7    "_id": "6376f1d7de48e5c61dfc3001",
8    "nombre": "ciencia ficcion",
9    "__v": 0
10 },
11 {
12   "_id": "6376faee1131f5ccb4f6a2f4",
13   "nombre": "romance",
14   "__v": 0
15 }
16 ]

```




POST

Items:

The screenshot shows a REST client interface in Visual Studio Code. The request is a POST to `http://localhost:3000/api/items` with a JSON body:

```

{
  "titulo": "Rambo",
  "tipo": "Pelicula",
  "genero": "Accion",
  "año": "1992",
  "duracion": 195,
  "created_at": "2022-11-16T00:00:00.000Z",
  "creado_por": "admin",
  "actores": [
    { nombre: "..." },
    { nombre: "..." }
  ]
}

```

The response is a 200 OK status with 13 bytes and a time of 41 ms. The response body is:

```

{
  "_id": ObjectId('63759004a4c7cbbb49df35c0'),
  "titulo": "Rambo",
  "tipo": "Pelicula",
  "genero": "Accion",
  "año": "1992",
  "duracion": 195,
  "actores": Array,
  "created_at": "2022-11-16T00:00:00.000+00:00",
  "creado_por": "admin",
  "_v": 0
}

```

The terminal shows the server is active on port 3000 and the connection is successful.

Actores:

The screenshot shows a REST client interface in Visual Studio Code. The request is a POST to `localhost:3000/api/actores` with a JSON body:

```

{
  "nombre": "Marlon Brandon"
}

```

The response is a 200 OK status with 17 bytes and a time of 33 ms. The response body is:

```

{
  "_id": ObjectId('6376193ba247388f0e600740'),
  "nombre": "Marlon Brandon",
  "_v": 0
}

```

The terminal shows the server is active on port 3000 and the connection is successful.

Géneros:

POST localhost:3000/api/generos

Query Headers² Auth Body¹ Tests Pre Run^{New}

Json Xml Text Form Form-encode GraphQL Binary

```
1 { "nombre": "ciencia ficcion" }
```

Status: 200 OK Size: 15 Bytes Time: 23 ms Response

1 Genero guardado

Preview

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** JUPYTER

```
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node index.js'
Servidor activo, puerto: 3000
Conexion Satisfactoria!
{}
{}
{ nombre: 'ciencia ficcion' }
```

PUT

Items:

PUT http://localhost:3000/api/items

Query Headers² Auth Body¹ Tests Pre Run^{New}

Json Xml Text Form Form-encode GraphQL Binary

Json Content Format

```
1 {
2   "id": "63759004a4c7cbbb49df35c0",
3   "titulo": "Rambo II"
4 }
```

Status: 200 OK Size: 26 Bytes Time: 25 ms

Response Headers⁶ Cookies Results Docs

```
1 {
2   "msj": "Item Actualizado"
3 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** JUPYTER

```
Conexion Satisfactoria!
{
  titulo: 'Rambo',
  tipo: 'Pelicula',
  genero: 'Accion',
  año: '1992',
  duracion: 195,
  created_at: '2022-11-16T00:00:00.000Z',
  creado_por: 'admin',
  actores: [ { nombre: '....' }, { nombre: '...' } ]
}
{ id: '63759004a4c7cbbb49df35c0', titulo: 'Rambo II' }
```

Object

```
{
  _id: ObjectId('63759004a4c7cbbb49df35c0')
  titulo: "Rambo II"
  tipo: "Pelicula"
  genero: "Accion"
  año: "1992"
  duracion: 195
  actores: Array
  created_at: 2022-11-16T00:00:00.000+00:00
  creado_por: "admin"
  __v: 0
}
```

24°C Nublado 8:53 p. m. 16/11/2022



DELETE

Items:

The screenshot shows a REST client interface in Visual Studio Code. The request is a DELETE to `http://localhost:3000/api/items/63759004a4c7cbbb49df35c0`. The response is `200 OK` with the message `1 item Eliminado con exito`. A terminal window at the bottom shows the server is running on port 3000 and the item was successfully deleted.

```
[nodeemon] restarting due to changes...
[nodeemon] starting 'node index.js'
Servidor activo, puerto: 3000
Conexion Satisfactoria!
eliminado: 63759004a4c7cbbb49df35c0
```

Actores:

The screenshot shows a REST client interface in Visual Studio Code. The request is a DELETE to `localhost:3000/api/actores/6376193ba247388f0e600740`. The response is `200 OK` with the message `1 actor eliminado con exito`. A terminal window at the bottom shows the server is running on port 3000 and the actor was successfully deleted.

```
Servidor activo, puerto: 3000
Conexion Satisfactoria!
{ id: '6376193ba247388f0e600740', nombre: 'Marlon Brando Jr' }
eliminado: 6376193ba247388f0e600740
```



Géneros:

DELETE
localhost:3000/api/generos/6376f189de48e5c61dfc2ffe
Send

Query
Headers²
Auth
Body¹
Tests
Pre Run New

Query Parameters

☐ parameter
value

Status: 200 OK
Size: 26 Bytes
Time: 10 ms

Response
Headers⁶
Cookies
Results
Docs

1 Genero eliminado con exito

Preview

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER

```

(nodemon) starting `node item.js index.js`
Servidor activo, puerto: 3000
Conexion Satisfactoria!
(nodemon) restarting due to changes...
(nodemon) starting `node item.js index.js`
Servidor activo, puerto: 3000
Conexion Satisfactoria!
eliminado: 6376f189de48e5c61dfc2ffe

```

Evidencia GitLab o GitHub

Evidencia de la realización de alguna actualización (commit), donde se visualice la actualización y el historial de actualizaciones (Versión)

Mel708 / EstoEsCine
Public
Watch 1
Fork 0
Star 0

Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights

main

Commits on Nov 14, 2022

update semana 3
nodeJS con el get de actores, items y generos. JSON de la base de datos en MONGO DB. nombre de base de datos
Mel708 committed 42 minutes ago

Commits on Nov 13, 2022

Add files via upload
Lo que se hizo en javascript relacionado al proyecto
daniel28893 committed 19 hours ago

Commits on Nov 1, 2022

creación carpeta documentos
sprint 1
Mel708 committed 13 days ago



main ▾ EstoEsCine / JSON base de datos /

Go to file Add file ▾ ...

Mel708 update semana 3 ...	d6e6401 44 minutes ago	History
..		
Captura.PNG	update semana 3	44 minutes ago
actores.json	update semana 3	44 minutes ago
generos.json	update semana 3	44 minutes ago
items.json	update semana 3	44 minutes ago
usuarios.json	update semana 3	44 minutes ago

Evidencia JIRA (Seguimiento del proyecto)

TABLERO Y ENLACE

<https://mgr708.atlassian.net/jira/software/projects/GG1PC/boards/5>

← → ↺ mgr708.atlassian.net/jira/software/projects/GG1PC/boards/5

Jira Software Tu trabajo ▾ Proyectos ▾ Filtros ▾ Paneles ▾ Personas ▾ Aplicaciones ▾ Crear

G04 Grupo 1 Proyecto ...
Proyecto de software

PLANIFICACIÓN

- Hoja de ruta
- Backlog
- Tablero**

DESARROLLO

- Código
- Páginas de proy...
- Añadir acceso rápido
- Configuración del pr...

Proyectos / G04 Grupo 1 Proyecto Ciclo4

Tablero Sprint 2

MR + Epic ▾

POR HACER 2 INCIDENCIAS

para el POST crear controlador, modelos y ruta para ITEMS, ACTORES y GENEROS

GG1PC-10

Visualizar consumo del API REST con Postman

GG1PC-11

EN CURSO 1 INCIDENCIA

para el GET crear controlador, modelos y ruta para ITEMS, ACTORES y GENEROS

GG1PC-9

LISTO 1 INCIDENCIA ✓

Crear base de datos en MONGO dB

GG1PC-12 ✓



Evidencias de las Reuniones de Equipo

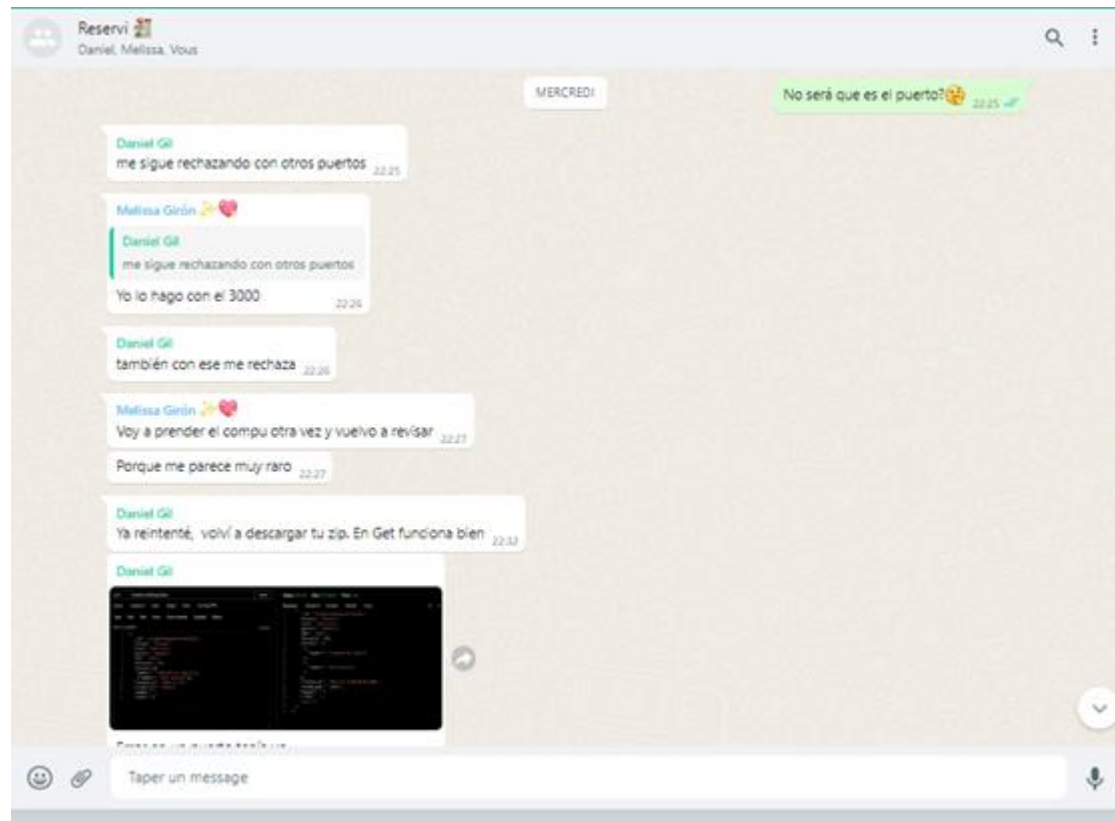
Como evidencia de las reuniones que efectúa el equipo del proyecto, presentar capturas de pantalla de las reuniones efectuadas y si lo consideran pertinente algunas actas de las reuniones.

REUNIÓN No. 02		
Fecha: 16 de noviembre 2022	Hora: 22:00	
Participantes: <ul style="list-style-type: none"> ➤ Daniel Fernando Gil García. ➤ Paula Andrea Gutiérrez Avendaño. ➤ Melissa Girón Rodríguez. 		

TEMAS ABORDADOS

Se dió solución a un problema que se tenía con la conexión con Thunder. Se actualiza el put y el delete de géneros.

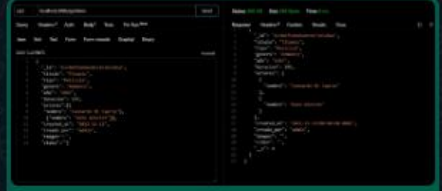
EVIDENCIA FOTOGRÁFICA



Voy a prender el compu otra vez y vuelvo a revisar 10:27 p. m.


Porque me parece muy raro 10:27 p. m.

Ya reintenté, volví a descargar tu zip. En Get funciona bien 10:32 p. m. ✓✓



Error en un puerto tenía yo 10:40 p. m. ✓✓

Melissa Girón
era lo del puerto 3000? 10:42 p. m.

Melissa Girón
 10:43 p. m.

Polis Programacion
Menos mal era eso 10:45 p. m.

Melissa Girón
sípi porque aca andba haciendo cosas y todo me salía :S 10:47 p. m.

sí '-' 10:42 p. m. ✓✓