UDACITY

Return to Classroom

DISCUSS ON STUDENT HUB

# UdaConnect

REVIEW

CODE REVIEW

HISTORY

## Meets Specifications

Even though the previous reviewer has passed the rubric of diagram, I would still encourage you to take a look at the suggestions and fix it accordingly.

Great job, you are ready to go! 💯 Clearly, you have acquired all the important concepts from this project. Wish you all the best for the upcoming projects! 👌
Tip: If you are interested in knowing more about kafka usage, I encourage you to read this introduction. Also, you could learn some valuable insights of using gRPC from reading this experience sharing

## Architecture Design

✓

Each module includes a 1-2 sentence justification for the module design choice either as a label on the design diagram or in a separate document

The justifications indicate that the decisions about the chosen protocols and technologies are based on business requirements including required scale. Your supervisor wants to be able to launch this project in 2 weeks with a limited budget. At the same time, the project needs to be able to scale to handle large volumes of location data being ingested.

The project is designed as an MVP and does not include any unnecessary features

- Cost and development time are minimized
- Services should run in containers
- Design should be able to handle ingress of a large volume of data

Good reasons are provided! The proposed architecture correctly reflects

✔ Services are correctly splitting into multiple services including

- Api
- Kafka consumer/producer
- Location api

✔ Design is able to handle ingress of a large volume of data by leveraging kafka and gRPC

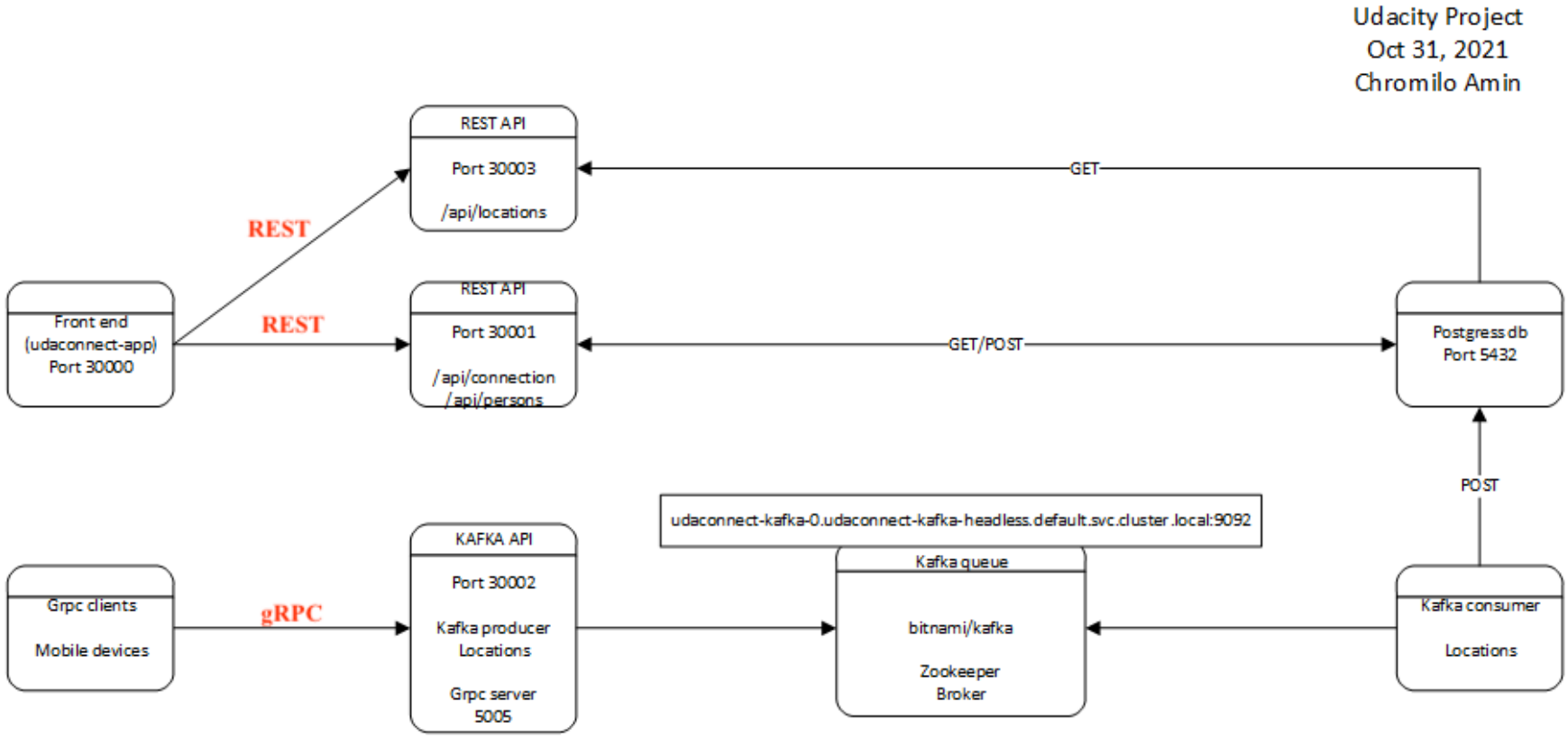
✓

Architecture diagram shows the design of the system as individual services. It should show the relationship between the frontend, API's, databases, and Kafka.

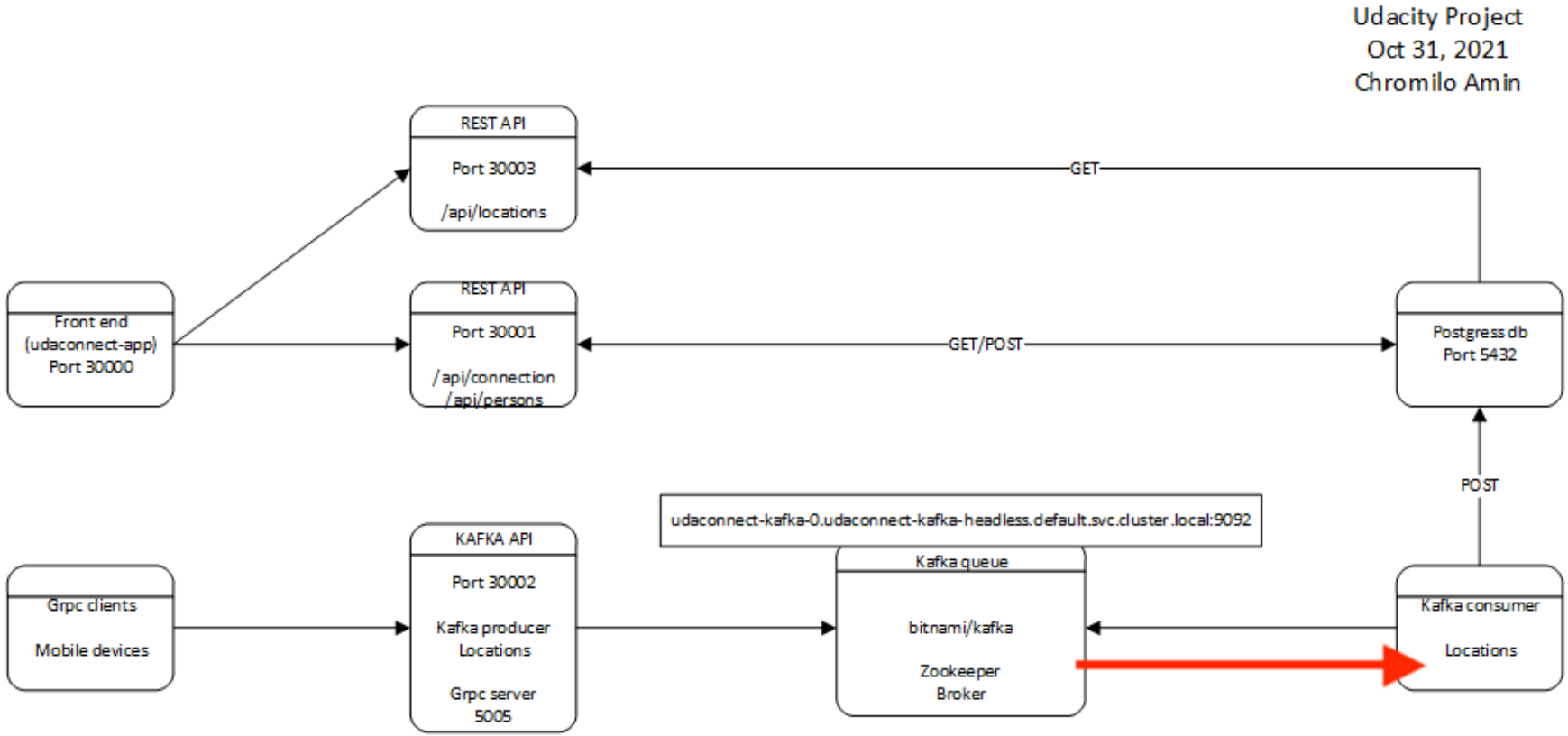
Arrows or lines connect the individual services to represent request/response relationships between services.

All of the necessary services and protocols are included as modules in the architectural design

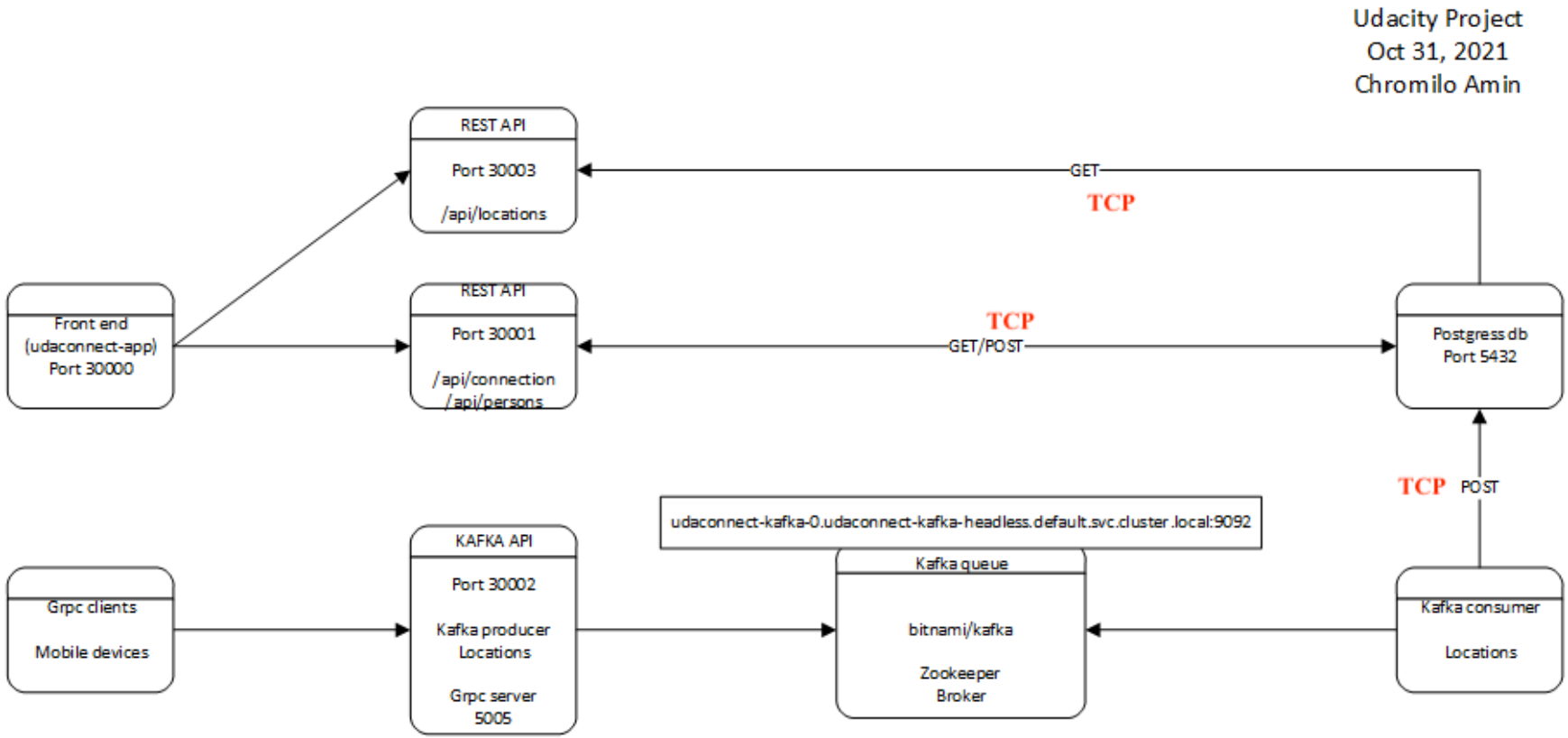
✔ The relationships between API and database are specified
✔ The relationship between API and kafka are specified
✘ Please specify the protocols between each service



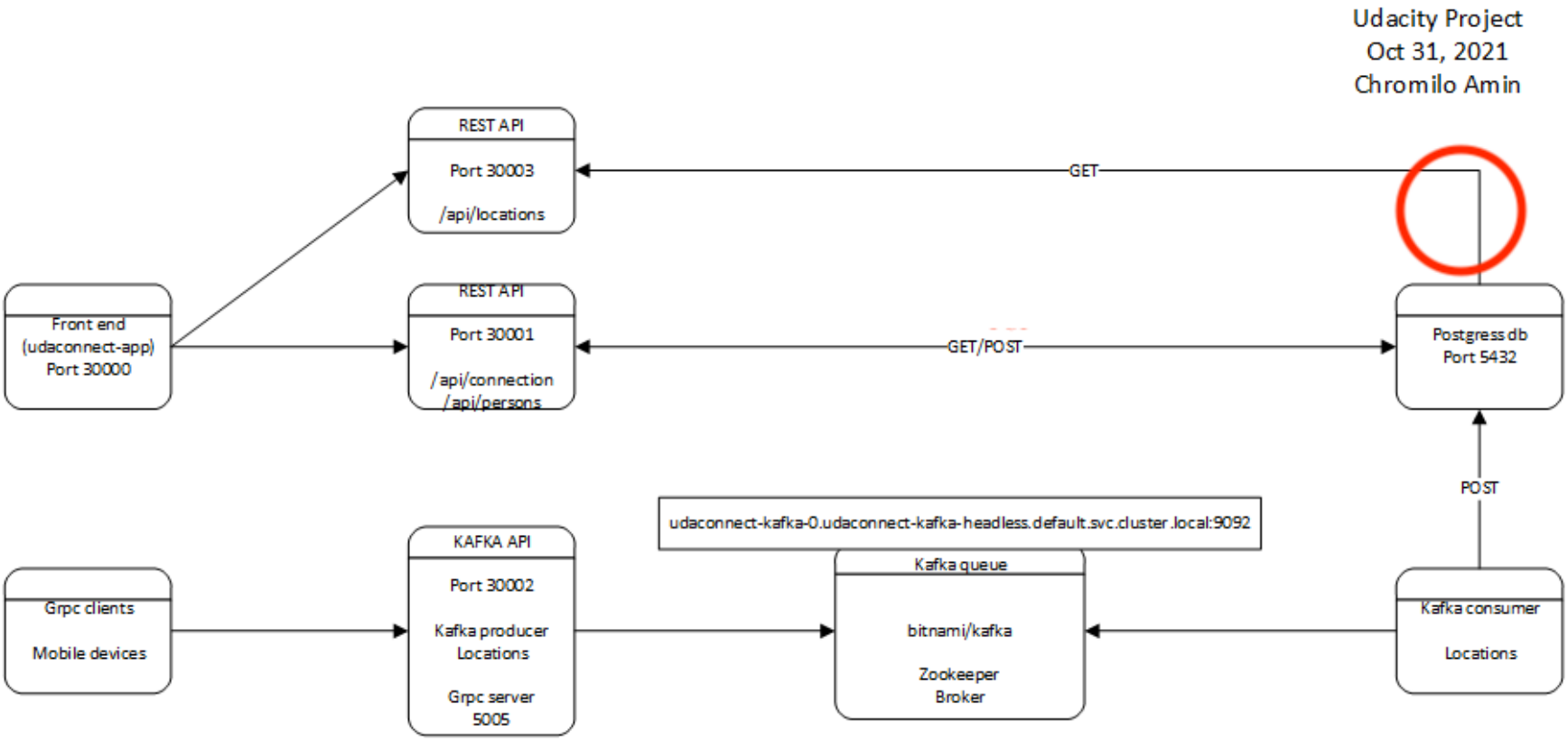

❌ Since a consumer gets the messages out from kafka, the arrow direction should be in the opposite direction




❌ The protocol between services and db is TCP instead of POST




❌ According to your location api implementation, the service can do both read/write operations to the db. So you should use bidirectional arrow instead of unidirectional arrow.


Udacity Project
Oct 31, 2021
Chromilo Amin
REST API
Port 30003
/api/locations
GET
REST API
Port 30001
/api/connection
/api/persons
GET/POST
Front end
(udaconnect-app)
Port 30000
Postgress db
Port 5432
POST
udaconnect-kafka-0.udaconnect-kafka-headless.default.svc.cluster.local:9092
KAFKA API
Port 30002
Kafka producer
Locations
Grpc server
5005
Kafka queue
bitnami/kafka
Zookeeper
Broker
Grpc clients
Mobile devices
Kafka consumer
Locations

```
@api.route("/locations")
@api.route("/locations/<location_id>")
@api.param("location_id", "Unique ID for a given Location", _in="query")
class LocationResource(Resource):
    @accepts(schema=LocationSchema)
    @responds(schema=LocationSchema)
    def post(self) -> Location:
        request.get_json()
        location: Location = LocationService.create(request.get_json())
        return location

    @responds(schema=LocationSchema)
    def get(self, location_id) -> Location:
        location: Location = LocationService.retrieve(location_id)
        return location
```

## Microservice Development

- Services have their own `Dockerfile`'s
- Services are deployable with Kubernetes through Kubernetes `Deployment`'s and `Service`'s

- Services are deployable with Kubernetes through Kubernetes `Deployment`s and `Service`s.

✔ Good job providing kafka consumer image
✔ Good job providing kafka producer image
✔ Good job providing frontend service image

✔ Good job providing location api image
✔ Good job providing api image

```
E:\Documents\udacity\SUSE\nd064-c2-message-passing-projects-starter\modules\location_api>kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
udaconnect-zookeeper-0                1/1     Running   2          3d1h
kafka-api-655c8d7859-jhpf6            1/1     Running   0          25h
postgres-5f676c995d-c5b9d             1/1     Running   15         42d
kafka-consumer-7d76dccc79-lnhv8       1/1     Running   3          11h
udaconnect-app-f7bd98cc5-sx852        1/1     Running   0          22h
udaconnect-kafka-0                    1/1     Running   6          3d1h
udaconnect-api-6b9949688b-fsl27       1/1     Running   0          8m19s
udaconnect-location-api-7f4c4bb6b6-p5sqn   1/1     Running   0          14s
```

✔

- Kubernetes deployments use Docker images built from the students' final solution and not from the starter code.

All the docker images are built from your solution, well done!

## Message Passing

✔

- Project implements a Kafka queue in a container in a `Dockerfile` or Kubernetes deployment file. This can be configured manually in a base image or using a pre-built Kafka Docker image.
- `requirements.txt` file installs the `kafka-python` Kafka library.
- `kafka` deployment runs successfully without errors.

✔ Kafka queue container is specified in Kubernetes deployment file

✔ The required kafka-python package is correctly specified in `requirements.txt`
✔ kafka can be deployed without any errors

✔

- Project contains a `*.proto` file showing that they mapped a message and service into a protobuf format. The `*.proto` file should have at least one `message` and one `service` declared in it.
- Code should contain a `*_pb2` and `*_pb2_grpc` file generated from the `*.proto` file.
- Project contains a gRPC client. If using Python with the standard `grpc` library, code should open a gPRC channel.
- Project contains a gPRC host. The standard `gprc` library code should contain a `grpc.server()` instantiation.
- `requirements.txt` file should define the `grpcio` package.

✔ Required `message` and `service` are implemented
✔ Required `pb2` files are correctly generated.
✔ Required `grpcio` is correctly defined in `requirements.txt`
✔ Required grpc channel is open
✔ Required grpc server is initialized

✔

- Project either created a new API endpoint(s) or a modification to existing Flask API.
- All new API endpoints use proper HTTP request types.
- `GET` or `DELETE` request does not contain an HTTP payload
- REST API's have live Swagger documentation in an external library (or manually written an OpenAPI spec.

✔ All API endpoints use proper HTTP request types.
✔ There is no payload needed for GET
✔ Required Swagger doc is provided.

# Udaconnect REST APIs 1.0.0 OAS3

Udaconnect Person, Connection, Location API

Contact the developer

Apache 2.0

**Servers**

https://virtserver.swaggerhub.com/chromilo/Udaconnect/1.0.0 - SwaggerHub API Auto Mocking

## users Open to all

POST `/api/persons` adds a person record

## udaconnect

GET `/api/persons` Retrieves Persons

GET `/api/persons/{person_id}` searches for a person

GET `/api/locations/{location_id}` searches locations

## Code Quality

Project runs without error

Code is appropriately documented. Comments are concise, relevant and not excessive

Code is neatly written with proper indenting. Python code follows PEP 8 guidelines.

- ✓ Project runs without error
- ✓ Code is clean and easy to follow

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START