TRIPTALLY

SC2006 SCSI GRP 5

Tan Wei Song Russell Bryan (U2421844K)
Justin Woon Thean Woon (U244641B)
Ethan Jared Chong Rui Zhi (U2421895B)
Evelyn Theresia Cuaca (U2320523L)
Parvez Kurniawan Wijaya (U2423845G)

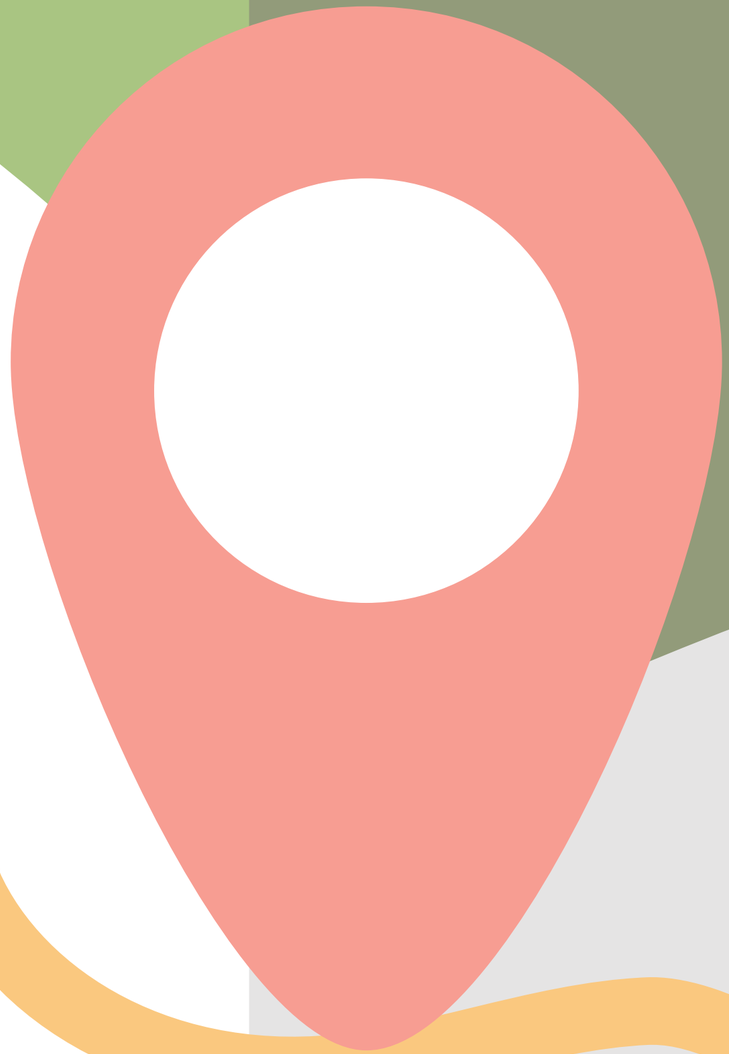


TABLE OF CONTENTS

INTRODUCTION

REAL WORLD CHALLENGES

Due to Singapore's high cost of living, "invisible" costs related to transportation pile up quick.

It is difficult to find parking lot availability, location and pricing in a centralised place for a chosen destination.

Limited map functionality available to display road hazards, closures, tolls and congestion for drivers to anticipate.

OUR SMART NATION'S INITIATIVE

Sustainability

- Singapore strives to be an eco-friendly, carbon-neutral nation through the use of sustainable technologies and initiatives.

Technology

- Singapore strives to be more efficient, productive and interconnected through the use of modern technologies.

OUR SOLUTION: TRIPTALLY

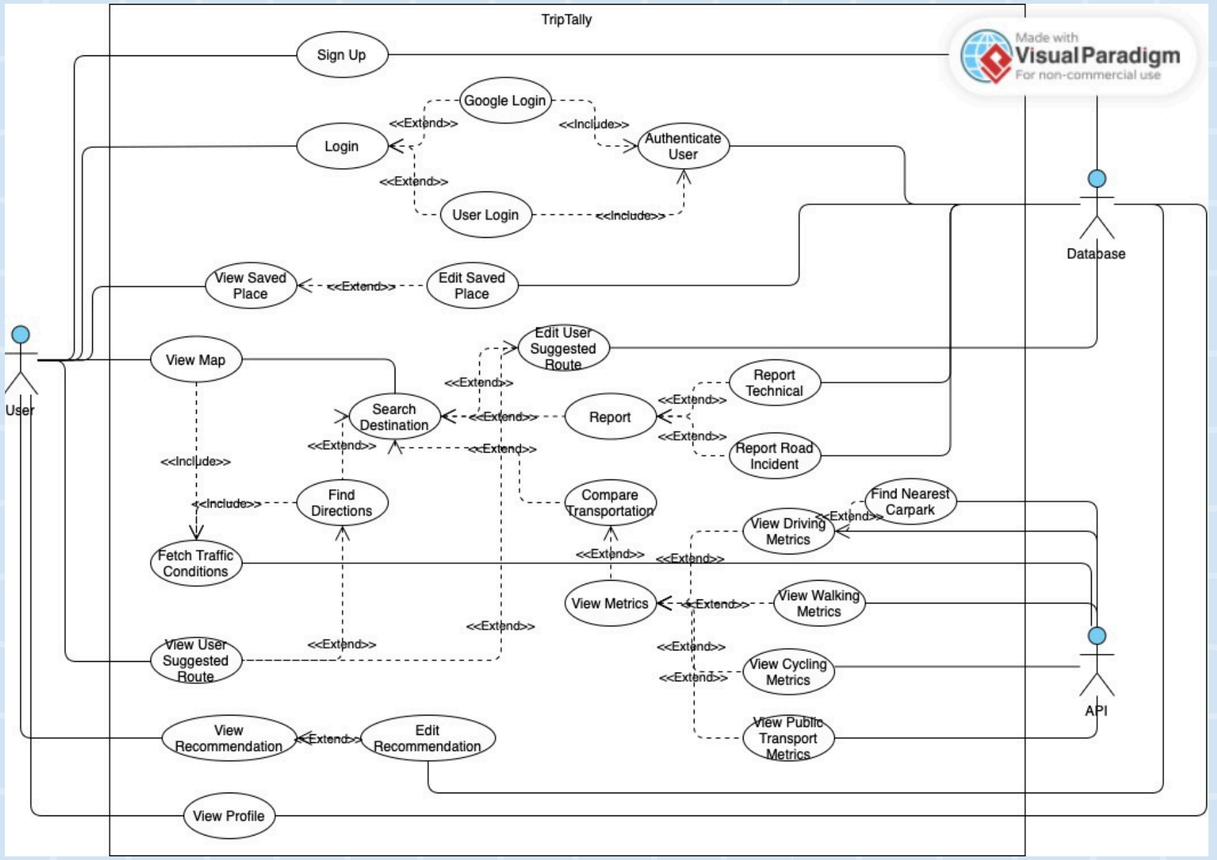
Sustainability

- Tracks CO_2 emissions from driving
- Tracks fuel consumption and fuel cost per trip
- Comparison tool to compare costs and emissions of different modes of transportation

Technology

- Interconnects drivers on the road through road hazard/incident reports and route sharing
- Collates government data such as
  - Carpark locations, availability and pricing
  - Public transport fares
  - ERP Charges and gantry locations
  - Traffic camera information

USE CASE DIAGRAM



TECH STACK USED

Front End

- Google OAuth
- React Navigation
- React Native
- Expo Go

Back End

- httpx
- PostgreSQL
- Ultralytics
- FastAPI
- Docker
- OpenCV
- Pydantic
- Redis
- Pandas

External APIs

- Google Cloud Platform
- TomTom Traffic API
- LTA DataMall
- data.gov.sg

data.gov.sg Datasets Used:

1. Bus fares for trunk, feeder and express buses (static csv, latest version always pulled from data.gov.sg)
2. MRT/LRT fares (static csv, latest version always pulled from data.gov.sg)
3. ERP gantry pricing (static pdf, manually converted to csv)
4. LTA gantry information (static geojson, latest version always pulled from data.gov.sg)
5. Traffic camera information (live API data fetched from data.gov.sg and refreshed in very short intervals)

SOLUTION (LIVE DEMO)

SYSTEM DESIGN

System Design

What we implemented ?

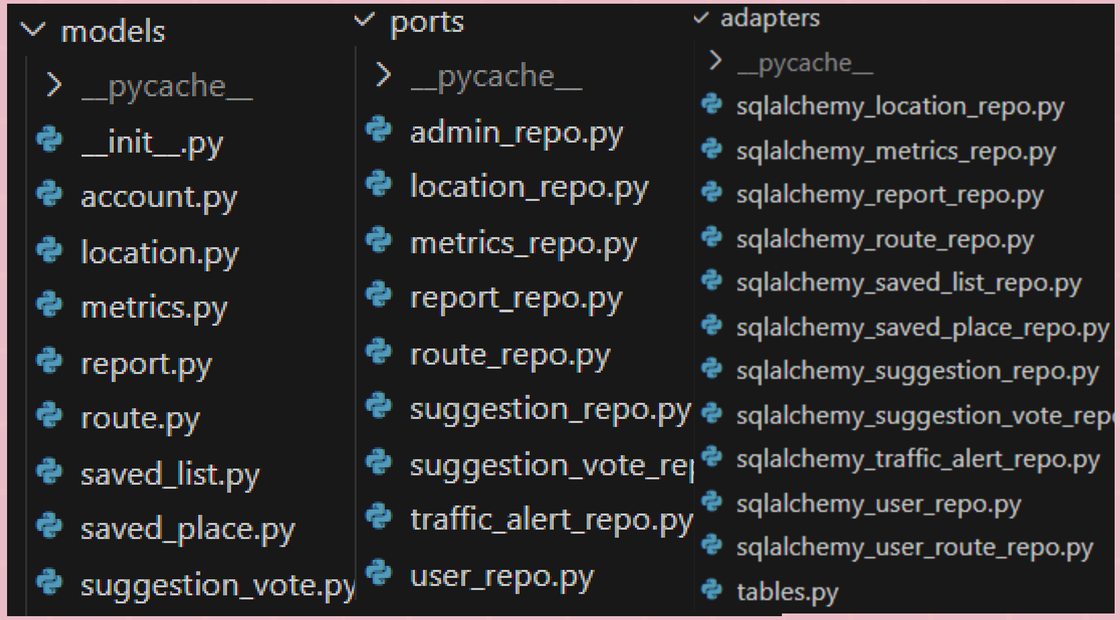
- Model-View-Controller (MVC)
- Extra abstraction (Service and Repository layers)

This make our app clean, testable, and modular

System Design

Model (M): Data Layer

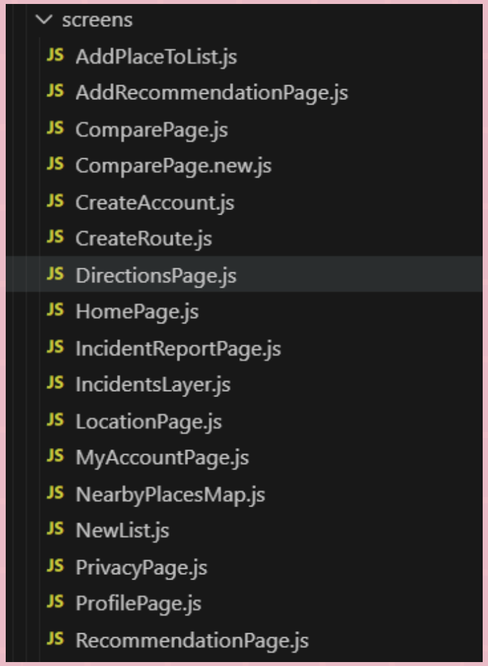
Location:
backend/app/models/,
backend/app/ports/,
backend/app/adapters/



System Design

View Layer (V): Presentation Layer

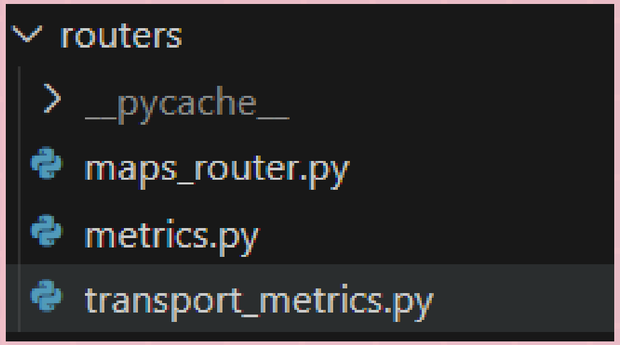
- Location : frontend/src
- Framework: React Native + Expo



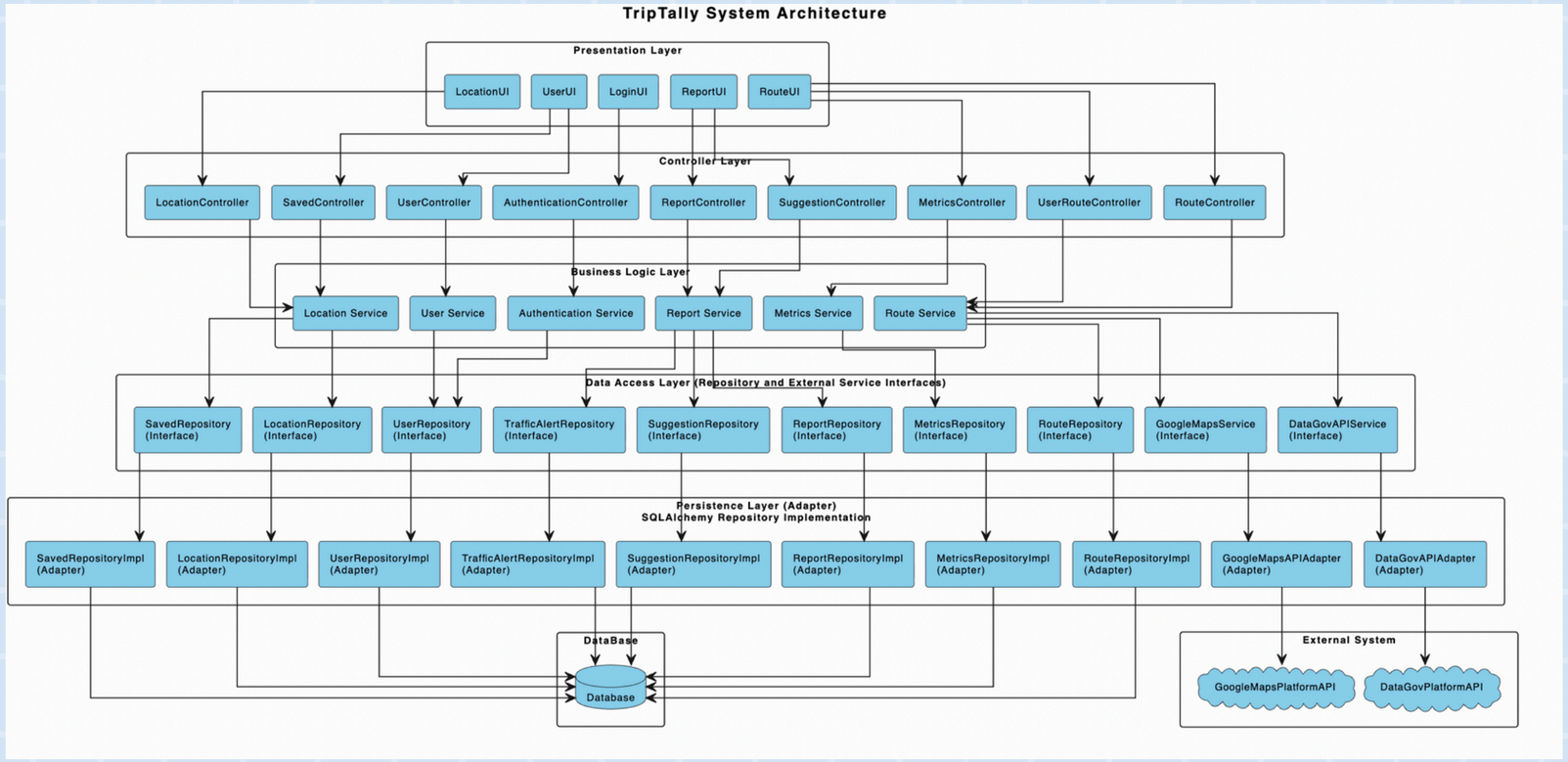
System Design

Controller Layer (C):

- Location : backend/app/routers/



SYSTEM ARCHITECTURE



DESIGN PATTERNS

Observer Pattern

Observer: UI

Subject: useState()

Subscribe using useEffect()

```
const [trafficAlerts, setTrafficAlerts] = useState([]);
const [tomtomIncidents, setTomtomIncidents] = useState([]);
const [selectedAlert, setSelectedAlert] = useState(null);
const [showModal, setShowModal] = useState(false);

useEffect(() => {
  const currentMarkerIds = new Set([
    ...trafficAlerts.map(alert => `alert-${alert.id}`),
    ...tomtomIncidents.map((feat, idx) => feat?.id || `tomtom-${idx}`)
  ]);

  Object.keys(markerAnimations.current).forEach(markerId => {
    if (!currentMarkerIds.has(markerId)) {
      delete markerAnimations.current[markerId];
    }
  });
}, [trafficAlerts, tomtomIncidents]);
```

DESIGN PATTERNS

Factory Pattern

Multiple possible types of camera repository

```python
class RepositoryFactory:
    """Factory for creating data repository instances"""

    @staticmethod
    def create(repo_type: Union[RepositoryType, str], config: Config = None) -> DataRepository:
        """Create repository instance based on type"""

        if repo_type == RepositoryType.REDIS:
            return RedisRepository(config.redis)

        elif repo_type == RepositoryType.CSV:
            return CSVRepository(base_dir=data_dir)

        elif repo_type in (RepositoryType.SQL, RepositoryType.SQLITE):
            return SQLRepository(db_path=db_path)
```

Multiple possible types of forecast algorithm

```python
class ForecasterFactory:
    """Factory for creating forecaster instances"""

    @staticmethod
    def create(forecaster_type: Union[ForecasterType, str], config: Config = None) -> ForecastingStrategy:
        """Create forecaster instance based on type"""

        if forecaster_type == ForecasterType.AUTO:
            # Try ML first, fallback to simple

        elif forecaster_type == ForecasterType.SIMPLE:
            return SimpleForecaster(max_history=max_history)

        elif forecaster_type in (ForecasterType.ML, ForecasterType.XGBOOST):
            return MLForecaster(model_dir=model_dir)
```

DESIGN PATTERNS

Repository Pattern

Camera repository can use Redis, SQL, or CSV

```python
class DataRepository(ABC):
    """Abstract base class for data repository implementations"""

    @abstractmethod
    def save_ci_state(self, state: CIState) -> bool:
        """Save current CI state"""
        pass
```

```python
class RedisRepository(DataRepository):
```

```python
class SQLRepository(DataRepository):
```

```python
class CSVRepository(DataRepository):
```

DESIGN PATTERNS

Strategy Pattern

```python
class ForecastingStrategy(ABC):
    """Abstract base class for forecasting strategies"""

    @abstractmethod
    def generate_forecast(self, state: CIState) -> CIForecast:
        """Generate forecast for given CI state"""
        pass

    @abstractmethod
    def get_strategy_name(self) -> str:
        """Return name of the strategy"""
        pass

    @abstractmethod
    def is_available(self) -> bool:
        """Check if strategy is available"""
        pass
```

Forecast algorithm (ML, simple) generated at runtime

```python
class MLForecaster(ForecastingStrategy):
```

```python
class SimpleForecaster(ForecastingStrategy):
```

DESIGN PATTERNS

Without Facade

```python
# User would need to:
config = Config.from_env()
repo = RedisRepository(config.redis)
geospatial = GeospatialService()
camera_loader = CameraDataLoader()
route = LineString(points)
cameras = geospatial.find_cameras_along_route(route, camera_loader.load_cameras(), radius)
# ... 20+ more lines of complex operations
```

With Facade

```python
class CIProcessingService:
    """Main service for CI calculation and forecasting"""

    def __init__(self, config: Config, repo_type: str = None, forecaster_type: str = None):
        # Initializes many complex subsystems:
        self.api_client = TrafficCameraAPIClient(config.api)
        self.ci_calculator = CICalculator(config.ci)
        self.motion_detector = MotionDetector(config.processing.cache_dir)
        self.context = ServiceContext.from_config(config, repo_type, forecaster_type)
        self.repository = self.context.repository
        self.forecaster = self.context.forecaster
        self.yolo = YOLO(...)
```

Simplifies route optimization pipeline

```python
# User just does:
optimizer = DepartureTimeOptimizer(repository, geospatial, camera_loader)
result = optimizer.find_optimal_departure(route_points, eta_minutes)
```

DESIGN PATTERNS

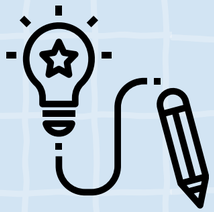
Singleton Pattern

```
camera_loader = get_camera_loader()
```

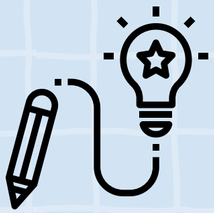
```
optimizer = DepartureTimeOptimizer(
    repository=repository,
    geospatial_service=geospatial,
    camera_loader=camera_loader
)
```

Decorator Pattern

```
@router.post("/optimize", response_model=OptimalDepartureResponse)
async def optimize_departure_time(request: OptimalDepartureRequest):
```



DESIGN PRINCIPLES



SINGLE RESPONSIBILITY PRINCIPLE

Where each module, class or file has one responsibility.

API Layer

Only handle http requests and delegate business logic to the service layer

```python
# app/api/user_routes.py
router = APIRouter(prefix="/users",

@router.post("/", response_model=Us
def create_user(user: UserCreate, c
    """Single responsibility: handl
    repo = SqlUserRepo(db)
    service = UserService(repo)
    # Delegates business logic to s
    return service.create_user(user
```

Metrics Calculation

Metrics files only serve the purpose of computing transport metrics; functions within are decomposed to ensure only one clear responsibility each.

```python
def calculate_fuel_cost(distance_km
    """Single responsibility: calcu
    fuel_used = distance_km * FUEL_
    cost = fuel_used * FUEL_PRICE_F
    return round(cost, 2)

def calculate_co2_emissions(distanc
    """Single responsibility: calcu
    emissions = distance_km * CO2_E
    return round(emissions, 2)
```

```python
def calculate_erp_charge(gantries_p
    """Single responsibility: calcu
    # ERP calculation logic only

def get_all_driving_metrics(distanc
    """Single responsibility: orche
    metrics = {
        "distance_km": round(distar
        "fuel_cost_sgd": calculate_
        "co2_emissions_kg": calcula
    }
    if polyline:
        metrics["erp_charges"] = ca
    return metrics
```

Service Layer

Service layer files only contain user-related business logic, does not access data.

```python
class UserService:
    def __init__(self, repo: UserRe
        self.repo = repo  # Depends

    def create_user(self, email: st
        """Create a new user with b
        # Single responsibility: us

    def update_user(self, user_id:
        """Update a user"""
        # Single responsibility: us
```

OPEN-CLOSED PRINCIPLE

Where code is open for extension but closed for modification.

Metrics Repo

Protocol definition is closed for modification:

```python
# app/ports/metrics_repo.py
class MetricsRepository(Protocol):
    def add(self, metrics: Metrics)
    def get_by_id(self, metrics_id:
    def get_by_route_id(self, route
    def list(self) -> list[Metrics]
    def update(self, metrics: Metri
    def delete(self, metrics_id: ir
```

But open to extension:

```python
# app/adapters/sqlalchemy_metrics_r
class SqlMetricsRepo(MetricsReposit
    """SQLAlchemy implementation"""
    def __init__(self, db: Session)
        self.db = db

    def add(self, metrics: Metrics)
        # SQLAlchemy-specific imple
        # ...
```

Metrics Class

Base metrics class is closed for modification:

```python
@dataclass
class Metrics:
    id: int
    total_cost: float = 0.0
    total_time_min: float = 0.0
    total_distance_km: float = 0.0
    carbon_kg: float = 0.0
    type: str = "metrics"
```

But open to extension:

```python
@dataclass
class DrivingMetrics(Metrics):
    fuel_usage_per_km: float = 0.0
    fuel_cost_per_liter: float = 0.
    fuel_liters: float = 0.0
    type: str = "driving"

@dataclass
class PTMetrics(Metrics):
    busFares: float = 0.0
    mrtFares: float = 0.0
    fares: float = 0.0
    type: str = "public_transport"
```

```python
@dataclass
class WalkingMetrics(Metrics):
    calories: float = 0.0
    type: str = "walking"

@dataclass
class CyclingMetrics(Metrics):
    calories: float = 0.0
    type: str = "cycling"
```

Reports Class

Base Reports class extendable into different report types

```python
# app/models/report.py
@dataclass
class Report:
    id: int
    user_id: Optional[int] = None
    time: Optional[datetime] = None
    status: str = "open"
    type: str = "report"
```

```python
@dataclass
class IncidentReport(Report):
    start_location_id: Optional[int
    end_location_id: Optional[int]
    obstruction_type: str = ""
    description: str = ""
    resolved: bool = False
    type: str = "incident"
```

```python
@dataclass
class TechnicalReport(Report):
    description: str = ""
    category: str = ""
    added_by: Optional[str] = None
    type: str = "technical"
```

LISKOV SUBSTITUTION PRINCIPLE

Where superclass objects can be substituted with its subclass objects without breaking the system.

Metrics Repo

Repository handles all types and works with Metrics but any subclass of Metrics can be substituted.

```python
# app/adapters/sqlalchemy_metrics_repo.py
class SqlMetricsRepo(MetricsRepository):
    def add(self, metrics: Metrics) -> Metrics:
        """Accepts ANY Metrics subtype - LSP compliant!"""
        if isinstance(metrics, DrivingMetrics):
            row = DrivingMetricsTable(...)
        elif isinstance(metrics, PTMetrics):
            row = PTMetricsTable(...)
        elif isinstance(metrics, WalkingMetrics):
            row = WalkingMetricsTable(...)
        elif isinstance(metrics, CyclingMetrics):
            row = CyclingMetricsTable(...)
        else:
            # Base Metrics also handled
            row = MetricsTable(...)
```

Report Class

Report Repository handles base Report type as well as derivative Report classes.

```python
# app/adapters/sqlalchemy_report_repo.py
class SqlReportRepo(ReportRepository):
    def add(self, report: Report) -> Report:
        """Accepts base Report or any subtype"""
        if isinstance(report, TechnicalReport):
            row = TechnicalReportTable(...)
        else:
            row = ReportTable(...)
```

Route Class

Base Route objects and subclass UserSuggestedRoute objects are interchangeable

```python
# app/models/route.py
@dataclass
class Route:
    id: int
    start_location_id: int
    end_location_id: int
    subtype: str
    transport_mode: str = ""
    route_line: list[int] = field(default_factory=list)
    metrics_id: Optional[int] = None
    type: str = "route"
```

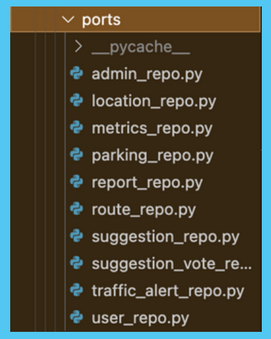
```python
@dataclass
class UserSuggestedRoute(Route):
    user_id: Optional[int] = None
    type: str = "user_suggested"
```

INTERFACE SEGREGATION PRINCIPLE

Where Clients should not have to depend on interfaces that they do not use.

Separate Repository Interfaces

Services working with different aspects of the program only access the repository(ies) they need.



Domain-Specific Methods

e.g. UserRepository only has User-specific methods that other Repositories do not contain.

```python
# app/ports/user_repo.py
class UserRepository(Protocol):
    # Standard CRUD
    def add(self, user: User) -> User: ...
    def get_by_id(self, user_id: int) -> Optional[User]: ...
    def get_by_email(self, email: str) -> Optional[User]: ...
    def get_by_username(self, username: str) -> Optional[User]: ...
    def list(self) -> list[User]: ...
    def update(self, user: User) -> User: ...
    def delete(self, user_id: int) -> bool: ...

    # User-specific operations
    def add_saved_location(self, user_id: int, location_id: int) -> bool: .
    def remove_saved_location(self, user_id: int, location_id: int) -> bool
    def get_saved_locations(self, user_id: int) -> list[int]: ...
```

Service Layer

Service Layer only uses required interfaces.
e.g. Route Service

```python
# app/services/route_service.py
def create_route(route_repo: RouteRepository, data: RouteCreate) -> Route:
    """Only depends on RouteRepository"""
    route = Route(...)
    return route_repo.add(route)

def list_routes_by_user(
    route_repo: RouteRepository,  # Only needs RouteRepository
    user_id: int
) -> list[UserSuggestedRoute]:
    return route_repo.list_by_user(user_id)
```

DEPENDENCY INVERSION PRINCIPLE

Where high-level modules do not depend on lower-level ones;
instead both rely on abstractions.

Protocol-based Repository Interfaces

Repository business logic is never dependent on our current protocol (SQLAlchemy). It can be interchanged with other database protocols.

```python
# backend/app/ports/metrics_repo.py
class MetricsRepository(Protocol):
    def add(self, metrics: Metrics) -> Metrics: ...
    def get_by_id(self, metrics_id: int) -> Optional[Metrics]: ...
    def get_by_route_id(self, route_id: int) -> Optional[Metrics]: ...
    def list(self) -> list[Metrics]: ...
    def update(self, metrics: Metrics) -> Metrics: ...
    def delete(self, metrics_id: int) -> bool: ...
```

```python
# backend/app/adapters/sqlalchemy_metrics_repo.py
class SqlMetricsRepo(MetricsRepository):
    def __init__(self, db: Session):
        self.db = db

    def add(self, metrics: Metrics) -> Metrics:
        # SQLAlchemy-specific implementation
```

Service Layer

Service Layer classes are dependent on Protocol interfaces (abstraction).

```python
# backend/app/services/user_service.py
class UserService:
    def __init__(self, repo: UserRepository):  # ← Depends on abstraction
        self.repo = repo

    def create_user(self, email: str, username: str, ...) -> User:
        user = User(...)
        return self.repo.add(user)  # ← Uses interface method
```

Service Functions

Service Functions take in Repository Interfaces as parameters (abstraction).

```python
# backend/app/services/route_service.py
def create_route(route_repo: RouteRepository, data: RouteCreate) -> Route:
    """Create a new route using any RouteRepository implementation."""
    route = Route(...)
    return route_repo.add(route)  # ← Works with any implementation
```

NON-FUNCTIONAL REQUIREMENTS

Requirement 1: Security

API keys stored in .env file
Password hashing and access token creation for user login
OAuth API usage for alternative login

```
GOOGLE_MAPS_API_KEY=[redacted]
DATABASE_URL=postgresql+psycopg2://postgres.[redacted]:triptally@aws-1-ap-southeast-2.pooler.supabase.
SECRET_KEY=mysecretkey
```

```python
def create_access_token(
    subject: str,
    expires_delta: Optional[timedelta] = None,
) -> str:
    if expires_delta is None:
        expires_delta = timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    expire = datetime.utcnow() + expires_delta
    payload: Dict[str, Any] = {"sub": subject, "exp": expire}
    return jwt.encode(payload, settings.SECRET_KEY, algorithm=ALGORITHM)


def decode_access_token(token: str) -> Dict[str, Any]:
    return jwt.decode(token, settings.SECRET_KEY, algorithms=[ALGORITHM])
```

```python
def hash_password(password: str) -> str:
    """Hash a password using bcrypt."""
    password_bytes = password.encode('utf-8')
    salt = bcrypt.gensalt()
    hashed = bcrypt.hashpw(password_bytes, salt)
    return hashed.decode('utf-8')


def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Verify a password against a hash."""
    password_bytes = plain_password.encode('utf-8')
    hashed_bytes = hashed_password.encode('utf-8')
    return bcrypt.checkpw(password_bytes, hashed_bytes)
```

Requirement 2: Performance

Debouncing for search and autocomplete and session token reuse reduces API calls

```
const debounceRef = useRef(null);

const handleSearch = (text) => {
  setQuery(text);
  clearTimeout(debounceRef.current);  // ← Cancel previous timeout

  if (!text || text.length < 2) {
    setResults([]);
    return;
  }

  // Wait 300ms before actually searching
  debounceRef.current = setTimeout(async () => {
    setLoading(true);
    // ... perform search
  }, 300);  // ← Only search after user stops typing
};

// Cleanup on unmount
useEffect(() => () => clearTimeout(debounceRef.current), []);
```

```
function AutocompleteBox({ placeholder, value, onChangeText, ... }) {
  const timer = useRef(null);
  const session = useRef(newToken());  // ← Reuse session across requests

  const query = (text) => {
    if (timer.current) clearTimeout(timer.current);
    onChangeText(text);

    if (!text || text.length < 2) {
      setResults([]);
      return;
    }

    // Debounce: wait 400ms after user stops typing
    timer.current = setTimeout(async () => {
      // ... fetch autocomplete results
    }, 400);
  };
}
```

Requirement 3: Reliability

Layer	Mechanisms	Benefit
Network	Retries, exponential backoff, timeouts	Handles transient failures
Validation	Input validation, coordinate checks, type safety	Prevents invalid data
Error Handling	Try-catch, finally blocks, detailed errors	Graceful failures
Database	Transactions, foreign keys, cascade deletes	Data integrity
UI	Optimistic updates, rollbacks, loading states	Responsive UX
Caching	TTL cache, request deduplication	Reduces load, faster responses
Monitoring	Health checks, logging, error tracking	Observability

```
async function request(path, { method = 'GET', body, token, timeout = 10000 } = {}) {
  const headers = {
    'Content-Type': 'application/json',
  };
  if (token) {
    headers.Authorization = `Bearer ${token}`;
  }

  const controller = new AbortController();
  const timeoutId = setTimeout(() => controller.abort(), timeout);

  try {
    const response = await fetch(`${API_BASE_URL}${path}`, {
      method,
      headers,
      body: body ? JSON.stringify(body) : undefined,
      signal: controller.signal,
    });

    clearTimeout(timeoutId);

    if (!response.ok) {
      let message = `Request failed with status ${response.status}`;
      try {
        const errorBody = await response.json();
        if (errorBody?.detail) {
          message = Array.isArray(errorBody.detail)
            ? errorBody.detail.map((d) => d.msg ?? d).join(', ')
            : errorBody.detail;
        }
      } catch {
        // ignore parse errors
      }
```

SOFTWARE TRACEABILITY

Use Case 2.3
SearchDestination

REQUIREMENT ELICITATION → REQUIREMENT ANALYSIS → DESIGN & IMPLEMENTATION-> TESTING

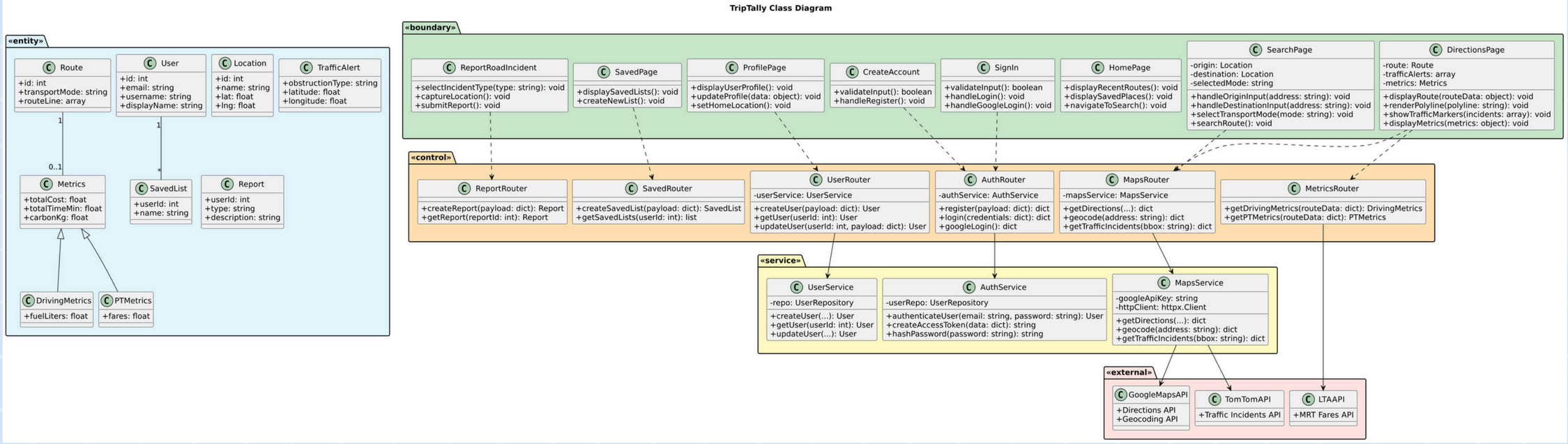
REQUIREMENT ELICITATION

Use Case ID:	#2-3		
Use Case Name:	SearchDestination		
Created By:	Russell Tan	Last Updated By:	Russell Tan
Date Created:	01/09/2025	Date Last Updated:	01/09/2025

Actor:	User, ExternalSystem
Description:	Displays the selected location to the user.
Preconditions:	1. Origin and Destination must be known. 2. Internet connection must be available.
Postconditions:	1. The selected location is displayed on the map for the user with the information regarding the location.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. System displays cached recent searches. 2. User keys in their destination in the search bar. 3. System queries geolocation API to search for the destination. 4. Location is displayed with its information and directions for the user. 5. Users can then select mode of transportation for the directions to their destination. 6. Route calculation is triggered. 7. Multiple routes to the users destination is displayed
Alternative Flows:	AF-1: Destination picked is from recent or Saved Places
Exceptions:	EX-1: Geocoding API is down. EX-2: Invalid location is inputted.
Includes:	1. ViewMap
Special Requirements:	None
Assumptions:	Interactive map is loaded.
Notes and Issues:	None

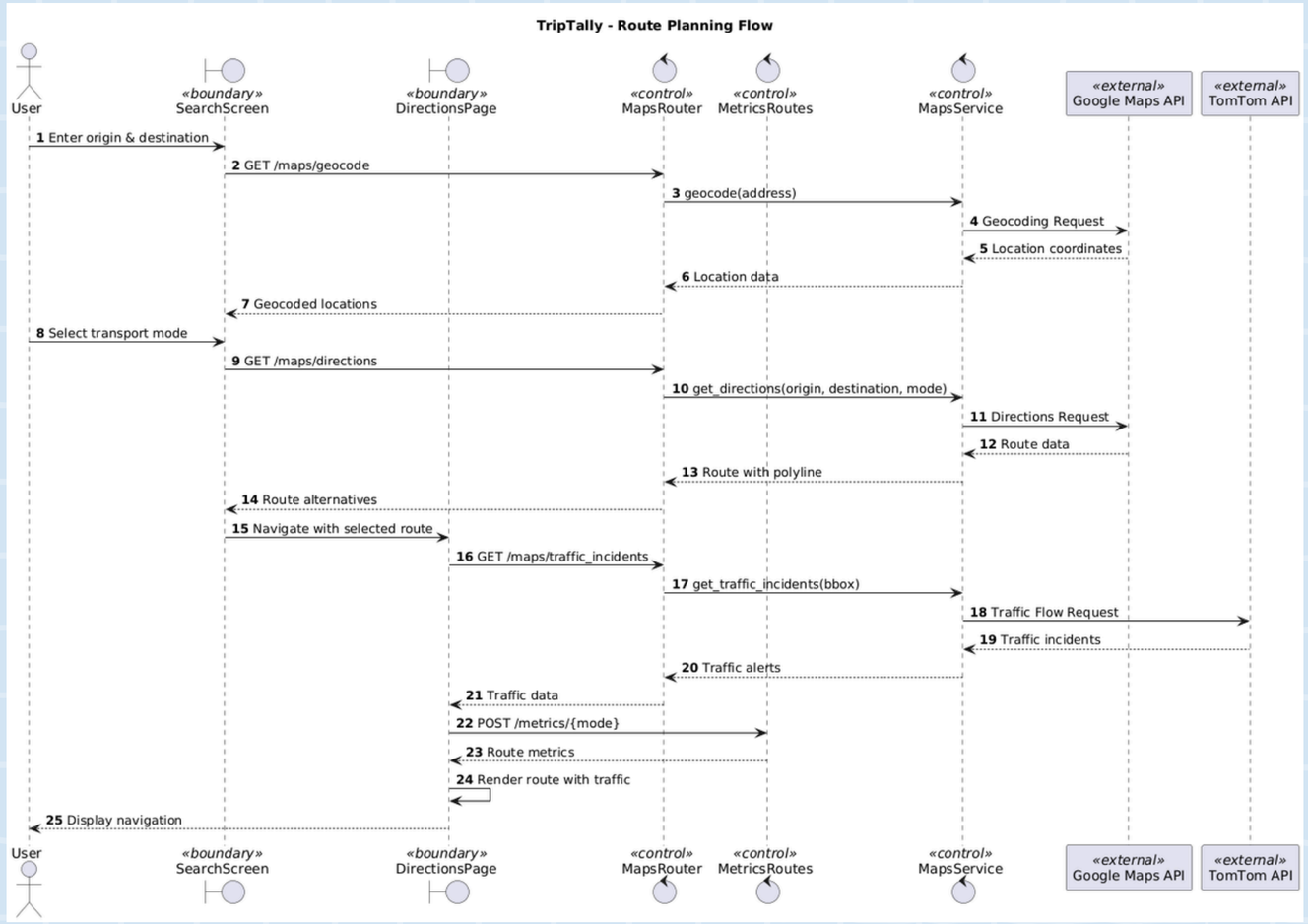
Use Case Description

REQUIREMENT ANALYSIS



Class Diagram

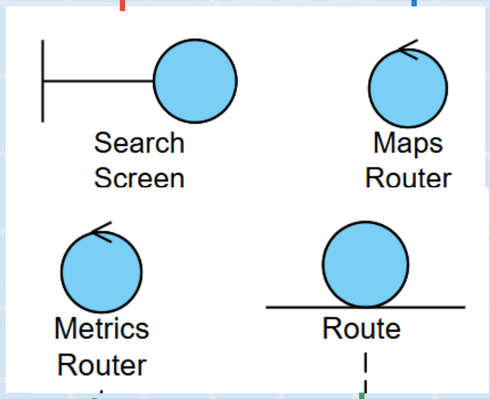
REQUIREMENT ANALYSIS

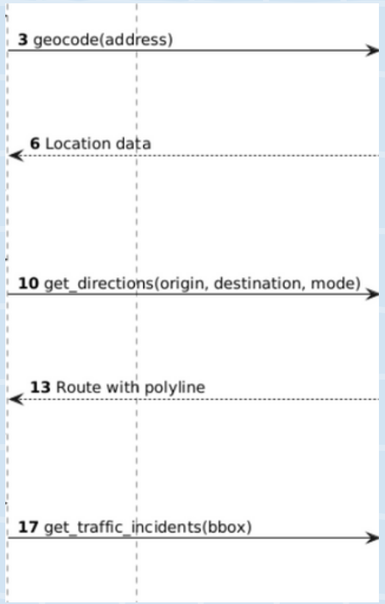


Sequence Diagram

DESIGN & IMPLEMENTATION

Separation of Concerns Modularity Traceability





Flow of Events:

1. System displays cached recent searches.
2. User keys in their destination in the search bar.
3. System queries geolocation API to search for the destination.
4. Location is displayed with its information and directions for the user.
5. Users can then select mode of transportation for the directions to their destination.
6. Route calculation is triggered.
7. Multiple routes to the users destination is displayed

TESTING

Black Box: Equivalence Class Testing

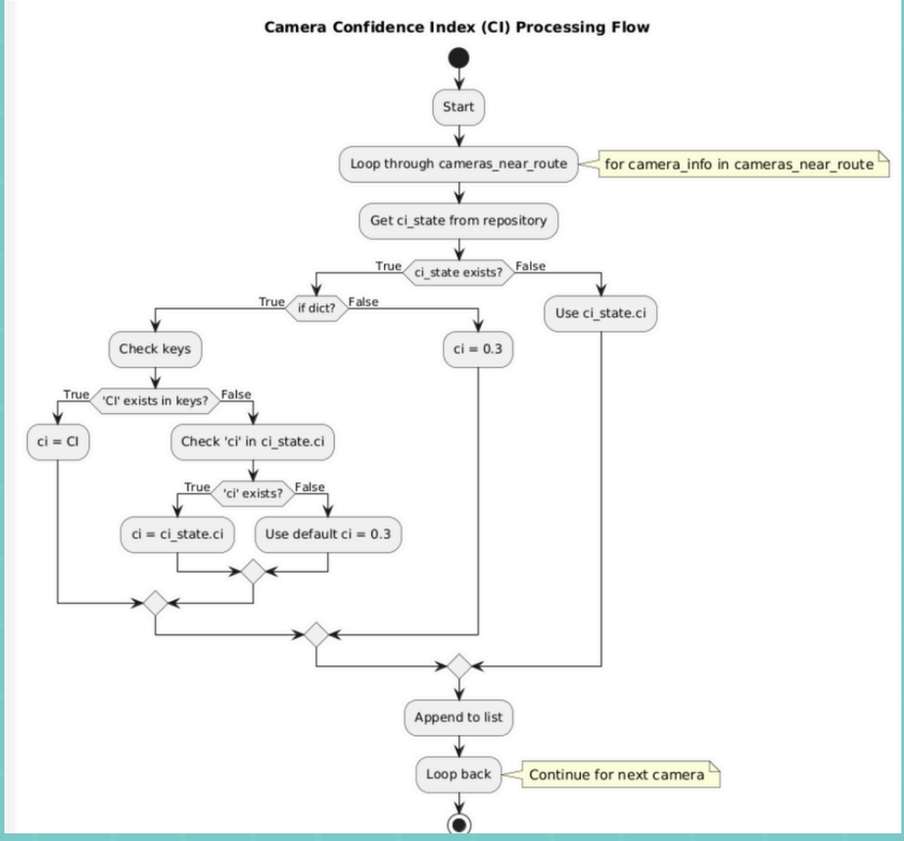
5. SearchPage Validation

2 < Valid Length < 200

Test ID	Input	Expected Output	Actual Output	Result	Description
	Location				
2.1	Valid Location	200: Display LocationPage	200: Display LocationPage	Pass	Valid Input
2.2	Invalid Location	400: Bad Request	400: Bad Request	Pass	Length too short
2.3	Invalid Location	400: Bad Request	400: Bad Request	Pass	Length too short
2.4	Invalid Location	400: Bad Request	400: Bad Request	Pass	Length too long
2.5	Invalid Location	404: Not Found	404: Not Found	Pass	Place does not exist
2.6	Invalid Location	400: Bad Request	400: Bad Request	Pass	Whitespace Input

TESTING Camera Confidence Index flow

White Box Testing



Cyclomatic Complexity = 5 + 1 = 6

TC	Camera state returned by repository	CFG branches hit	Expected CI per camera	Expected `avg_ci`
T1 (zero-iter)	`cameras_near_route = []`	loop not entered ⇒ zero-iteration path	—	**0.0**
T2 (dict, 'CI' valid)	`{"CI": 0.65}`	dict? → yes → `'CI' in keys?` → yes (valid)	**0.65**	**0.65**
T3 (dict, 'CI' present but None)	`{"CI": None}`	dict? → yes → `'CI' in keys?` → yes but invalid ⇒ fall through to default	**0.3**	**0.3**
T4 (dict, only 'ci' valid)	`{"ci": 0.42}`	dict? → yes → `'CI' in keys?` → no ⇒ `'ci' in dict?` → yes (valid)	**0.42**	**0.42**
T5 (dict, neither key usable)	`{}` (or `{"ci": None}`)	dict? → yes → `'CI' in keys?` → no ⇒ `'ci' in dict?` → no/invalid ⇒ default	**0.3**	**0.3**
T6 (object path + loop-back)	two cameras: `[CIState(ci=0.80), {}]`	object? → use `ci_state.ci`; second camera goes to default; covers loop back/append	**[0.80, 0.30]**	**0.55**

FURTHER IMPROVMENTS

FURTHER IMPROVEMENTS

Push Notifications and Alerts

- Alerts users on traffic incidents which are within the vicinity of their location
- Users can toggle this feature on or off

Smart Route Suggestions

- Use Machine Learning based recommendations for best routes based on user's historical patterns, time of day and current traffic

Integrating Bike Rental Services

- Linking maps to see availability of rental bikes such has AnyWheel
- Users able to reserve a bike in advance

Multi Factor Authentication

- Addition of 2FA by sending OTP to the linked email or phone number for added security

Thankyou