

**TAPS**

**(Transport and Passenger Services)**

**Software Requirement Specification (SRS)**

# Executive Summary

This document describes the desired properties and features (“requirements”) for a suite of reusable, production quality software that implements on the TAPS project. The document is mainly intended for guiding architecture and implementation.

As final algorithm and parameter selection is a continuous process, the actual algorithms and their precise properties are not defined in this document, nor is the architecture of the final software suite.

The software suite can be targeted for both resource-constrained embedded platforms and high performance multi-core architectures. The software is mainly written in performance-optimized PHP language, and is mainly self-contained in the sense that few external libraries are required. Assembly language optimizations can be added to “bottleneck” positions but all functionality must be available in PHP. We require the code to be written in such a way that it is easily maintainable for at least 5 years.

The Application Programming Interface (API) can be required for calculating the actual charge of every Vehicle or Calculating the Kilometers. It should need the open cage geo location API that will help to get the Latitude and Longitude of a particular place.

This is a high-assurance software development project. During development, emphasis is on quality and correctness of code. An automated test system is continuously maintained for all components of the software suite.

The software need to integrate a payment gateway of final order transaction and it should be of Razor pay payment gateway.

**Table of Contents**

1 **Introduction** ..........................................................................................................................1

1.1 Purpose and Goals.....................................................................................................................2

2 **Functional Requirements**.......................................................................................................3

2.1 Background................................................................................................................................4 2.2 Functionality ..............................................................................................................................5 2.3 Assumptions and Dependencies .............................................................................................6 2.4 Application Programming Interfaces.......................................................................................7

3 **Nonfunctional Requirements**.........................................................

3.1 Secure Implementation Techniques........................................................................................8 3.2 Performance optimization.......................................................................................................8 3.3 Licensing ..................................................................................................................................9

**1 Introduction**

* 1. **Purpose and Goals**

This document describes the desired properties and features (“requirements”) for a suite of reusable, production quality software that implements on TAPS project. We refer to the resulting open source software deliverables simply as the “software suite”.

In addition to the desired functional goals of this suite, we define a uniform programming style and methodology that allows the software suite to meet the needs of the various case studies and anticipated applications.

We define the goals and priorities of the software developed within this project as follows:

1. **Correctness**. The software must implement its algorithms faithfully for all allowed inputs. A set of automatic scripts, test vectors and other data must be available for all components.
2. **Readability and documentation**. The code must be clear and understandable, and written in good, consistent style. All components must be clearly documented within the code. Programming interfaces and tools should also be externally documented.

1. **Flexibility**. The software is designed for maximum modularity and allows easy configuration. It must be possible to create libraries that contain strictly only those components that is required for a given application or target platform.
2. **Code security.** Code must be robust enough so that it does not form a “weak link” in an application that utilizes the TAPS functionality provided by the software. Strict security practices for writing, auditing, and verifying code must be adhered to from the start
3. **Development methods.** In addition to general secure coding practices, TAPS software requires special considerations to prevent leakage of confidential information in timing and physical attacks.
4. **Integration.** The effort required to integrate the software with existing applications that utilize others algorithms should be minimum. Furthermore, where standard Application Programming Interfaces exists for TAPS project, these should be used.
5. **Portability.** All functionality and much of the code must work on a wide spectrum of target platforms, from low-end microcontrollers to high performance server systems. Target specific optimizations should be made to core components (or bottlenecks) where they gained speedup or other advantage is high.
6. **Performance.** Algorithms must be implemented efficiently. For many arithmetic subtasks multiple algorithms are available; the project generally always implements the algorithm with best asymptotic complexity.
7. **Lifespan and maintainability.** Tools and techniques should be chosen in a way that one can expect the code to be easily maintainable for at least 5 years. Tools that experience frequent interface or syntax changes should be avoided.

**2 Functional Requirements**

**2.1 Background**

We define a set of software requirements for both resource-constrained embedded platforms and high performance multi-core architectures for use within the TAPS project and beyond. In the context of the TAPS project, the objective of Work Package 6 is to develop a suite of software routines implementing the constructions identified during WP4 to address various use cases. However, the development of these constructions and their parameter selection is an ongoing process. Our purpose is to produce a highly portable, flexible, efficient, general framework TAPS that we expect to have a long lifetime and high industry impact.

**2.2 Functionality**

The primarily function of the software is to booking Taxi as the requirement of the customers. They can book their needy vehicle any time with TAPS. TAPS have all types of vehicles it can be passenger vehicle as well as all ranges of loaders from mini to heavy loaders.

Customers have to give their pickup location and destination place and we will calculate the Kilometers between the places so that we can calculate the actual charge for every vehicle on TAPS and show it to our customers

After then they will go to book their order firstly they have to login itself after then they can proceed further.

Once it will login to their TAPS account they will be redirect to the bill page where they can see their invoice bill on this invoice bill here a terminology called Taps-Money it is rewarded price for the referral. For every referral customer will get Rs 150 taps money and it will be 10% executable in every order of the customer.

Once they checkout they will be redirect on the payment page where they have all the option for payment like UPI, Pay later, Card, Net Banking etc.

Finally we will get all the details of the order placed.

If anyone wants to attach their vehicle as a taxi they will have a separate area on the software called “Saathi Registration” They only have to give their details on the software.

If anyone wants to get the franchise of the company they will have a separate area on the software called “Franchise” They only have to give their details on the software.

Software will also have a seprate section for corporate tie-up where they only need to fill their details and our company will get back in 2 business days.

**2.5 Assumptions and Dependencies**

Implementation work is dependent on the availability and stability of appropriate TAPS algorithms. Implementation work primarily focuses on new methods and algorithms that may reach standardized status, and their support functions.

**2.6 Application Programming Interfaces**

Our software uses the **Geo-location API** from Open cage platform which help us to get the Latitude and longitude of the given place which helps us to calculate the kilometers between two places.

Our software also uses the API for payment gateway integration to our software which is **Razor pay**.

**3 Nonfunctional Requirements**

**3.1 Secure Implementation Techniques**

TAPS software requires special implementation techniques. Some of the basic requirements are:

1. Compare secret strings in constant time 2. Avoid branching controlled by secret data 3. Avoid table look-ups indexed by secret data 4. Avoid secret-dependent loop bounds 5. Prevent compiler interference with security-critical operations 6. Prevent confusion between secure and insecure APIs 7. Use separate types for secret and non-secret information 8. Use separate types for different types of 9. Clean memory of secret data

**3.2 Performance optimization**

Performance optimization should be performed on a best effort basis. Software should be no more than 15% slower than best-known implementation for any given algorithm. In algorithm parameter selection, security is the overriding priority before performance.

Asymptotically best algorithms are used for tasks such as ring arithmetic and parameter compression.

1. **Correctness.** A mathematically correct plain C implementation must always be available
2. **Performance**. We are generally interested in the maximum performance of the scheme.
3. **Implementation size.** Implementation footprint is especially important for the embedded targets; sometimes it may be necessary to sacrifice some performance in order to make the algorithm usable on low-end targets.