Ledger, A Real-Time Scheduling Application

Software Requirements Specification

Version 1.0

Authors:

Aaron Carrasco,
Keondre Credit,
Trevor Chaney,
Eduardo Dominguez

Prepared as per instruction by:

Rodion Podorozhny

For CS 4398 Software Engineering Project, Texas State University

Last revised: 17 July, 2020

Table of Contents

Preface

1) Introduction

The purpose of this document is to provide insight into the requirements of the Ledger scheduling application. The document will outline the system and illustrate how different components of the system are going to interact with each other, the user, and the system administrators. The document will give insight to potential customers and developers in the first deployment process of the application.

2) Purpose

The purpose of this system requirement specification is to set forth the requirements of Ledger in order to ensure its success as a scheduling application that will streamline the process of making appointments, schedules, and maintaining them between users and businesses.

2.1) Problem statement

Why another scheduling application? The problem with scheduling is there are many variables to account for in regards to when, where, how, and with what or whom the task should be accomplished. Who can attend to a customer, who cannot? By building Ledger we intend to account for those variables and to accommodate future variables by creating an easily adaptive system to meet the complex scheduling needs of our clients.

3) Definitions

Term	Definition
Application	Computer software system. When used in this document, it refers to specifically the Ledger scheduling system.
API	Application Programming Interfaces, or APIs, are an abstraction of software functions into a service provided to interact with said software.
Appointment	The designation of a time and/or place at which an event will occur.
Frontend	Used to refer to the client's side interface of the application.
Backend	Used to refer to the server side interface and programming interface of the application.
User	An individual who is interacting with the system as either a customer or an employee.
Customer	An agent that interacts with the application to initiate a scheduling process.
Employee	An agent that is assigned a schedule.
Schedule	A list of events and the times and/or places at which they will occur.
Client/Admin	An organization or individual working on behalf of an organization that is employing the use of the application.

Figure 1.1) Definition table.

4) Scope/Requirements

The following describes the functions of the Ledger application.

User Side:
- create an account
- view available time slots

- make an appointment
- manage appointments made (cancel, edit, move)

Admin Side:
- define available dates
- edit available dates
- manage appointment for all users
- delete account

Server Side:
- send confirmation notice appointments made
- send reminders of upcoming appointments
- model users and appointments

Comprehensive Description

1) Constraints and Assumptions

1.1) List of Assumptions

Ledger is built as a scheduling app that will be employed by our clients. Which means they will already have their own systems. The scheduling app will work as an auxiliary application. For our clients it is assumed that they will be working through a desktop so their interface will function through a web browser. While the users side will be able to schedule their appointments through a mobile/desktop browser interface.

2) List of Dependencies

2.1) Frontend

The dependencies for the frontend to function are as follows:
- React <16.13.1>
- React-DOM <16.13.1>
- Formik <2.1.5>
- React-Bootstrap <1.2.1>

All react libraries are to facilitate the functionality of ReactJS, a user interface library created by facebook. The formik library is used for input validation as the user is logging into the system.

2.2) Backend

The dependencies for the backend to function are as follows:

- Django <3.0.8>
- Django Rest Framework <3.11.0>
- Djoser <2.0.3>
- Django CORS headers <3.4.0>

3) Design Constraints

Due to the constraints of time in the development of this project we will be focusing on making the Ledger application functional on a web based platform first and foremost and, time permitting, we will then develop a mobile interface which our customers can use on their phones and tablets so that they can view their schedules on the go without needing to be at a desktop computer.

Functional Requirements

1) Summary

The Ledger application will begin by prompting the user to log into the application or create a new account through the “sign up” section. After which it will be determined what privileges they should have. Depending on what privileges the account is granted by the client, the user will be able to block out times, generate a work schedule, create recurring appointments, create times of unavailability, etc…

To facilitate this functionality, the following tools and technologies will be employed:

- Django, a python web framework to administer and program the backend.
- React, a javascript library for dynamically rendered components for the user interface.

- Git and GitHub, a distributed version control system for tracking changes made to files.

2) Descriptions of Operation

2.1) All users must login.

Existing Users will go through a login phase to access scheduler

Description : An existing customer with an account already setup is trying to log into our scheduling system.

Actors :

1. The user attempting to log into their account.
2. Ledger System

Flow of Events :

- Basic Flow
1. Input correct password and username, both are verified and the user is taken to their account page.
2. User incorrect password or username, page gives user notice to retry login attempt.
- Alternate Flow
1. After 3 failed attempts the user is taken to a reset password page. This will prompt the user to reset password with different verification questions.

Preconditions :

- The user must have already created an account.

Exit Conditions :

- Success Guarantee : Once a member is logged in the user can click “LogOut” to exit the account.
- Minimum Guarantee : Entering the correct username and password.

Figure 3.1) Description of user login process.

2.2) Customers make appointments

As a customer of a business, the customer should be able to make appointments.

Description : A logged in customer makes an appointment for a specific date and time.

Actors :

1. Customer Making Appointment
2. Ledger System

Flow of Events :

- Basic Flow
1. Customer Clicks on "Make Appointment Button"
2. Customer selects specific date
3. A view of available time slots is shown
4. Customer selects time
5. Clicks "Make Appointment"
- Alternate Flow
1. Customer Clicks on "Make Appointment Button"
2. Customer selects specific date
3. A view of available time slots is shown
4. Customer selects time
5. Time is unavailable customer; customer is asked to to select another time
6. Customer selects available time slot
7. Clicks Make Appointment

Preconditions :

- A customer must have an account and be logged in.
- A customer cannot have conflicting appointments

Exit Conditions :

- Success Guarantee : The customer is taken to home page and receives a confirmation notice to email
- Minimum Guarantee : Selecting date and time that is marked available on the calendar
- Success Guarantee : Administrator can see the status of the user now as employee
- Minimum Guarantee : Changing status of employee and saving change

Figure 3.2) Description of the process of a user creating an appointment in the Ledger System.

2.3) Admin privileges

Administrator can grant employee permissions to existing users

Description : Administrator grants permissions/access to employees

Actors :

1. Administrator
2. Ledger System

Flow of Events :

- Basic Flow
1. Administrator clicks on specific user account
2. Administrator is able to view the user account's information
3. Administrator changes the status permission from regular user to employee
4. Administrator saves changes
- Alternate Flow
1. N/A

Preconditions :

- User already has a profile/account made

Exit Conditions :

- Success Guarantee : Administrator can see the status of the user now as employee
- Minimum Guarantee : Changing status of employee and saving change

Figure 2.3) Description of the administrative privilege to grant other accounts elevated privileges.

2.4) Employee schedules

Employee updates schedule of available dates

Description : Employee updates the schedule to change some available dates for appointments; in this scenario the employee is blocking dates for scheduling preventing schedules to be made.

Actors :

1. Employee
2. Ledger System

Flow of Events :

- Basic Flow

1. Employee clicks on “edit schedule”
2. Employee clicks on “Blocks Dates”
3. Warning stating no appointments can be made on selected time frame
4. Employee confirms the changes
5. Schedule is update to omit selected dates

- Alternate Flow

1. Employee clicks on “edit schedule”
2. Employee clicks on “Blocks Dates”
3. Selected time frame already has appointment cannot done until those appointments are cancelled
4. Employee fixes dates to block off / edits current appointments
5. Schedule is update to omit selected dates

Preconditions :

- User already has a profile/account made and schedule is already made

Exit Conditions :

- Success Guarantee : Calendar view now has time frame as blocked off
- Minimum Guarantee : no customer can make appointments for those dates.

Figure 3.4) Description of the process of an employee updating their availability.

2.5) Customer preferences

Making Appointments with select employees

Description : A customer has the option to choose the employee they would like to make their appointment with.

Actors :

1. Customer
2. Ledger System

Flow of Events :

- Basic Flow
1. Customer selects “make appointment”
2. Checks available time slots
3. Choose their desired time slot checks for their desired employee they would like to meet with
4. The employee is available for set time slot
5. Customer selects “Make/Confirm appointment”
- Alternate Flow
1. Customer selects “make appointment”
2. Checks available time slots
3. The Employee they will like too meet is not available
4. Customer clicks “search dates by employee”
5. Customer select date and time when employee is available
6. Customer select “make/ confirm appointments”

Preconditions :

- User already has a profile/account made and schedule is with available time slots for appointments

Exit Conditions :

- Success Guarantee : customer has successfully made an appointment with the desired employee
- Minimum Guarantee : customer should see if the employee they request is available on the shown slots

Figure 3.5) Description of the ability for a customer to select an employee preference.

Non-Functional Requirements

It is paramount that a scheduling system be robust and always available to the user. With the investment of the livelihoods of our customers in our hands it behooves us to put the utmost care in crafting the most fault tolerant and reliable system possible.

1) Reliability

Knowing that a system is reliable is a part of building trust in a system. To insure that our customers can trust that the system will be there when they need it, there will be a local backup of the appointment database so that, though the application is primarily based in a web browser, they will be able to access appointment information even if their internet connection is disrupted.

2) Robustness

To develop a robust system that is also able to perform complex scheduling there is a need for easy to navigate and use documentation. A system can be crushed under its own complexity even if it's structure is sound. Not only will Ledger be soundly crafted it will be complex and thus we will provide a well written technical documentation that is also user friendly.

3) Performance

In regards to performance, a right answer delivered too late is a wrong answer in scheduling tasks. There must be an efficient deliverance of schedules given to employees as close to real time as possible. This is why Ledger will use a dynamic hashing algorithm to provide the best scheduling available to our customers.

4) Maintainability

By isolating the app components we can create a maintainable product. Each component and file will have its own responsibility and utility. This will allow any developer to fix any particular component without having to parse through the entire application's components. This will be done by creating a folder structure of the app that isolates the components as explained above. The final product will be more detailed however a general view of our implementation can be seen here:

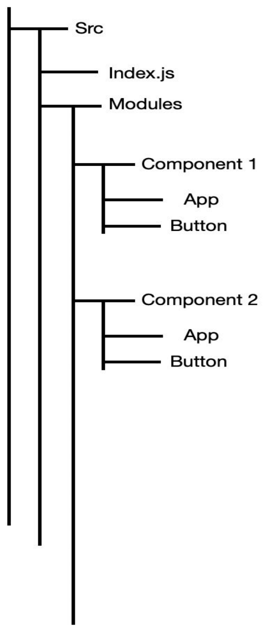
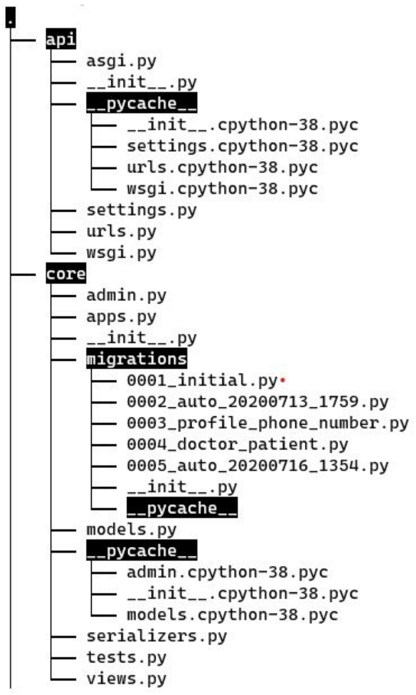


Figure 4.1) File structure diagram of project source code for frontend.



```
.
├── api
│   ├── asgi.py
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-38.pyc
│   │   ├── settings.cpython-38.pyc
│   │   ├── urls.cpython-38.pyc
│   │   └── wsgi.cpython-38.pyc
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── core
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   │   ├── 0001_initial.py
│   │   ├── 0002_auto_20200713_1759.py
│   │   ├── 0003_profile_phone_number.py
│   │   ├── 0004_doctor_patient.py
│   │   ├── 0005_auto_20200716_1354.py
│   │   ├── __init__.py
│   │   └── __pycache__
│   ├── models.py
│   ├── __pycache__
│   │   ├── admin.cpython-38.pyc
│   │   ├── __init__.cpython-38.pyc
│   │   └── models.cpython-38.pyc
│   ├── serializers.py
│   ├── tests.py
│   └── views.py
```

Figure 4.2) File structure diagram of project source code for backend.

5) Usability

But again, what good is a system if no one is able to use it. What is worse if there is a steep learning curve to that usability. We will provide tutorials and easy to understand and parse documentation to provide the swiftest understanding of the Ledger system so that our customers and their employees can forget about scheduling and focus on getting things done.

References

Below you will find a list of reference material related to the tools and documentation for the Ledger project.

The Ledger Github contains the most current source code of the project. "https://github.com/CS4398-Capstone-Project/Ledger".

The ReactJs Documentation was used for the development of the client, customer, and employee facing interfaces. "https://reactjs.org/docs/getting-started.html".

The Django Documentation, used in the development of the server side API and services. "https://docs.djangoproject.com/en/3.0/".

The Django REST framework Documentation, used in the development of the server side API and services. "https://www.django-rest-framework.org/".