MVP PRD: PocketCampus

Date: 2011-05-32

Overview

PocketCampus is an all-in-one mobile application to enhance the university experience for EPFL students. It centralizes various aspects of campus life, from academic schedules to dining options, transportation, and campus events. The app aims to streamline day-to-day activities, making campus studies and life on campus more efficient and interactive.

The app is extensible to support the all-in-one model. New services can be added easily into the app over time. The services fall into two broad categories: (1) those that do not require any authentication and provide convenient access to EPFL information directly on the phone, and (2) those that require authentication, and for which PocketCampus acts as a trusted mediator of the personalized information to deliver to the phone. All authenticated services are enabled via the EPFL single sign-on service (SSO).

The app is free for all students, who are the primary persona for this app. Perhaps later, other personas will be added as additional targets (e.g., professors, or laboratory assistants).

Pragmatically, we plan to develop this app ourselves, with minimal technological dependencies on EPFL's IT infrastructure (other than OAuth2 for single sign-on, which has been agreed upon).

The goal of the MVP is to build a minimal viable app that achieves significant penetration within the EPFL student body, in terms of aggregate downloads, monthly active users, and number of users that have trusted the application with their EPFL SSO credentials.

Based on informal conversations with EPFL management, we plan to reach with this MVP a user threshold that will make the app "*interesting*" to EPFL, so that we can enter into a first commercial contact.

Our model is to operate PocketCampus as a "mobile software-as-a-service" with support for both Android and iOS.

History

The proof-of-concept (POC) built during the semester had only basic features (Food, People directory, Public transit schedule). This included a mix of authenticated and unauthenticated features that demonstrates usefulness of the app.

The implementation of the PoC is totally hardwired; each feature consists of a different module within the Android application. For a solution like PocketCampus to be successful, we need to rethink the product and design it with extensibility in mind. Therefore, the MVP requires a change in the architecture.

Analysis of the situation

Currently, all EPFL services are managed separately. Therefore, as a student at EPFL, I have to go to one portal to see my grades, a different one to find out the menu, and yet another one to look at events happening on campus or access the library.

None of these services are available on a mobile phone; in fact, most EPFL websites do not support modern standards of web responsiveness, and are very difficult to access from a mobile device. This makes navigating campus life at EPFL very hard and time-consuming for me as I am already overburdened with academic load.

The value proposition

To remedy the situation, PocketCampus offers an all-in-one mobile app to access campus resources. This creates incentive for return users since they no longer have to go to multiple portals and login multiple times to access resources. The app developed as a POC, i.e., during the class project, resonated with students, who will be the primary target user base.

While students are the users, the EPFL administration is the customer. For EPFL as the customer, PocketCampus offers the following value proposition:

- As a leading technological institution, this provides EPFL with a modern, mobile-only all-in-one platform that is very broadly used and deployed among the student population, with the ultimate goal to have most students be monthly active users of the application.
- The extensible design enables a jointly agreed-upon feature roadmap, including features that only make sense on a mobile phone (e.g. that depend on localization) and features that replace mature software (e.g., IS-Academia for the release of grades, etc). This design will provide EPFL management the flexibility to deploy new services on the mobile platform.
- PocketCampus operates the service as a turn-key solution; it handles break fixes resulting from planned and unplanned changes in the EPFL websites during the year according to an annual service-level agreement.

PocketCampus offers a clear benefit to EPFL because it addresses a key pain point of the student experience.

PocketCampus is delivered as “mobile software as a service” to EPFL. This arrangement is convenient to EPFL, which currently lacks the technical expertise and human resources to operate a mobile service. It provides a way to get started.

As the relevance of PocketCampus grows, we need to ensure that we are also relevant and important to the EPFL IT technical teams, so that content is ideally exposed via API rather than having to scrape constantly-moving websites.

By operating the service, we will gain insights into which features are most useful and relevant, and gather metrics that will facilitate the renewal of the contract.

The pricing model is to-be-determined, based on discussion with EPFL IT, it might be:

- An annual commitment for the operation of the service itself (support cost, etc)
- (if needed) an additional commitment to support some particularly expensive features that require constant updates.
- A development fee for new services based on a jointly-approved roadmap.

The MVP

The PocketCampus MVP aims to bring a few of EPFL's most used resources under one roof to meet the value proposition identified above. High user retention will serve as a key metric to landing EPFL administration as a customer. We identify ICPs (Ideal Customer Profiles) below and list the set of features that we think are used enough to generate repeat traffic.

Concretely, we aim to integrate the following EPFL services into PocketCampus:

1. **Food Menu** - This is the easiest to do and is appealing to all users. Additionally, if a user is signed in we can highlight their meal price instead of the guest price.
2. **Printing** - an out-of-the-box printer configuration brings massive convenience to users. Provides an easy way to submit print jobs to EPFL's myPrint service.
3. **Moodle** - Tracking and managing course deadlines and accessing course content is the primary function of every student and teaching staff at EPFL.
4. **Camipro** - As a seamless method of payment, accepted by all retailers on EPFL campus, managing camipro balance is an excellent feature to simplify.
5. **News** - Integrating the EPFL news bulletin into the MVP enhances the visibility of EPFL's achievements. Moreover, being an authless feature, it can be shown to visitors and non-EPFL members as well.

Personas and Scenarios

User Stories for all EPFL Citizens (students, employees, professors)

1. "As an EPFL citizen, I want to easily access the campus food menu to make informed meal choices."

2. “As an EPFL citizen, I want to be able to quickly send documents from printing right from my phone to EPFL’s myPrint service in a few clicks.”
3. “As an EPFL citizen, I want to easily be able to view and recharge my camipro account within a few clicks.”

User Stories for Students only

1. “As a student at EPFL, I want to easily view my class schedule so that I can manage my time efficiently.”
2. “As a student, I want to view course announcements, handouts, and deadlines all in one place on my phone.”

User Stories for Professors (later)

1. “As a professor at EPFL, I want my students to promptly follow updates on Moodle to ensure they are well informed and up-to-date with course requirements.”

User Stories for EPFL Visitors (later)

1. “As a parent of a prospective student, I want to easily access the latest news and updates about EPFL’s activities and achievements, so that I can stay informed about the new and exciting developments at the university, helping me understand the environment and opportunities it offers to its students.”

Success Criteria

User Adoption and Engagement

- Target to onboard at least 10% of the campus population (1,000 users) within the first month of launch.
- Achieve a DAU of 10% (100 users) and MAU of 70% (700 users) in the same month.
- Achieve a retention rate of at least 60% in the following month.

Feedback and Satisfaction

- Achieve a minimum average rating of 4 out of 5 stars on app stores within the first two months of launch.
- Conduct monthly surveys to gather user feedback, aiming for at least 80% positive feedback regarding app usefulness and user experience.
- Implement at least half of the user-requested features or improvements within two weeks following each feedback survey. Promptly responding to user requests not only demonstrates our commitment to continuously enhancing the user experience but also significantly contributes to building and maintaining user trust in our platform.

Commitment from EPFL IT

- Engage with EPFL IT at the executive level without delay
- Discuss terms for first contact before the user adoption criteria are met, conditional on their achievement.

Features out (Roadmap)

Exclusion of ISA Integration The MVP of PocketCampus will not include integration with EPFL's ISA grade management system. This decision is primarily due to concerns about overloading the ISA service, as simultaneous access of grades from both ISA and our app could lead to extraneous traffic and system strain. This may prompt EPFL to blacklist our service. Additionally, integrating sensitive data requires a comprehensive compliance framework for data privacy and security in agreement with EPFL administration, which is beyond the scope of our initial rollout.

Exclusion of Parking Services Parking services will be excluded from the MVP due to the potentially dynamic nature of EPFL's parking policies. Immediate and accurate updates for parking are crucial to avoid misinformation, which could potentially lead to fines or other issues for users. To maintain the app's reliability and reputation, we have decided not to include parking services until we can ensure real-time accuracy in reflecting policy changes.

Non-functional requirements

We document the non-functional requirements that the MVP must satisfy to make it *viable.*

Security, privacy, and data retention policies

Security and Trust

EPFL services are guarded by Tequila (which is EPFL's OAuth2.0 system). Since our servers will be communicating with EPFL servers on the user's behalf, we must ensure that we do not save any user credentials on the backend (such as passwords or refresh tokens).

Privacy Considerations

As we plan to support printing, our server should only serve as a job submitting middleware and not save files of users since they could be private. Moreover, our backend should be so designed that we should not save/extract or infer any personally identifiable information about a user, such as Name, Age, Gender, Address, etc.

Data Retention Policy

In the eyes of our real customer (EPFL IT), we should be viewed simply as an information facilitator. This means that we need to keep a strict and well-defined policy around what user data we retain. For example, in the context of the food menu, our app could be viewed as favoring some cafeterias/food trucks over others if we collect and retain all the menus and user preferences to send them targeted ads. Moreover, for Moodle, we should not retain course assignments, handouts, and other material beyond the semester. Therefore, we should define clear data retention policies, like deleting food menus older than 30 days.

Adoption, Scalability, and Availability

Adoption

1. The application must, by default, have all push notifications disabled to avoid unnecessary digital distractions for the users.
2. Users should have the option to selectively enable push notifications for specific functionalities within the app, such as low camipro balance.
3. The application's user interface must include easily accessible settings for customizing notification preferences.
4. Adoption metrics must focus on user engagement quality rather than frequency, emphasizing the app's value-add to the users' experience.

Scalability

1. The application must be capable of scaling resources automatically to accommodate fluctuating user loads, particularly during peak academic periods.
2. The system should maintain optimal performance and response times, even under high traffic conditions, as defined by peak usage metrics (lunch time, exam period, etc.).
3. Scalability solutions must include database read-replicas and backend load balancing capabilities to ensure resource efficiency and service continuity.

Availability

1. The application must achieve a minimum uptime of 99.9%, ensuring consistent availability to users.
2. Implement comprehensive failover mechanisms to handle potential system disruptions, including power, network, and hardware failures.
3. Regular data backups and a robust disaster recovery plan must be in place to enable quick restoration of services in case of system failures.
4. Continuous API monitoring systems like sentry must be employed to detect and address performance issues and crashes proactively.

Functional requirements

Over-the-air (OTA) updates using "Plugins"

Motivation The PocketCampus MVP aims to integrate four critical EPFL campus services as its core functionality: food menus, Moodle, Camipro, and printing services, and one non-critical publicity feature : EPFL news. These services are hosted on separate EPFL platforms and are managed and updated independently. The app must be capable of handling asynchronous updates to these APIs without frequent client-side updates. This integration is essential to ensure that users have seamless access to the various services. Therefore, the application's design must include mechanisms to seamlessly connect with and display data from them despite their individual update schedules and API changes.

Proposed Solution To mitigate the challenges of direct API calls and frequent app updates, we propose:

1. A dynamic front-end app that queries an API to fetch details about the app view.
2. A backend server that integrates with EPFL APIs and can be updated as they update.

This design choice circumvents the need for immediate app updates in response to changes in EPFL's APIs, thus avoiding downtime related to Playstore/AppStore approval processes. By adopting an Over-The-Air (OTA) update model akin to those used in the tourism and travel industry, our app ensures uninterrupted access to EPFL services. This not only enhances user experience but also facilitates the incorporation of new features and updates, keeping the app current and responsive to the evolving needs of the EPFL community.

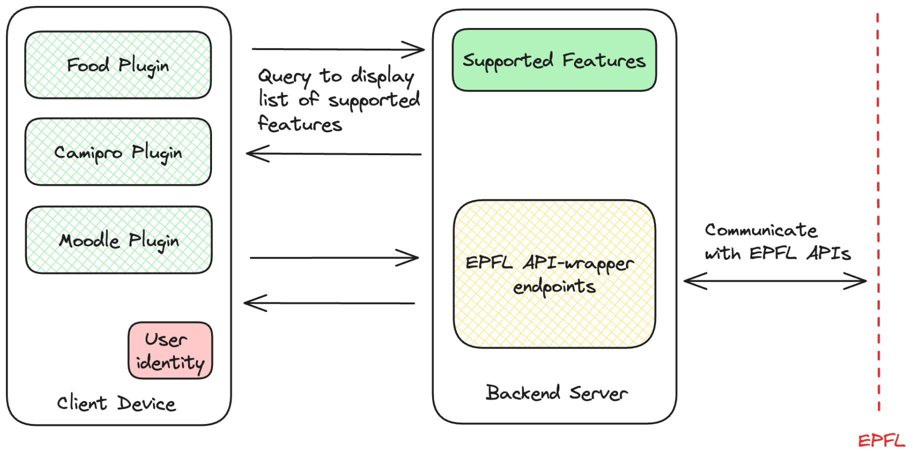


Fig1. OTA Update Model

Proactive Data Scraping and Caching

Motivation The 'Data-Prefetching' functionality is a critical component of our application, especially for features such as food menus and EPFL news. Both these services have slow data update frequency – for the food menu, it's one per day, and for the news, it is event-based. We wish to prefetch and cache this data on our servers for the following two reasons:

1. Enhanced Speed and Reliability: By prefetching and locally storing data such as daily menus and news, we reduce the app's dependency on real-time queries to EPFL's servers. This approach ensures faster data retrieval, providing users with immediate access to the latest information, thereby enhancing the overall user experience.
2. Load Management and Compliance: Frequent queries to EPFL APIs from a single origin can lead to throttling or flagging of our service as a security risk. By prefetching data, we minimize the frequency of our API calls, reducing the risk of being perceived as a malicious or compromised endpoint by EPFL's IT security systems.

Proposed Solution To manage this, we will implement a system to prefetch and cache this data on our backend servers. For the food menu feature, this means fetching upcoming menus in advance, and for news, it involves running a cron job that checks for updates several times a day. The prefetching process will be automated using a backend cron job. This job will run at regular, strategically determined intervals to ensure that the data stored on our servers is always current and synchronized with EPFL's latest updates.

Note that this cached data is subject to our privacy and data-retention policies - we do not cache any private user data like camipro balance, assignments etc. All cached data comes from publicly accessible EPFL services. Moreover, we don't store food menus for over 30 days.

User analytics and acceptance

Before full rollout of the MVP, we must implement and put in place a pipeline to collect relevant information on how users are using our application. This will allow us to identify improvements and fix bugs. Additionally, we will track the following metrics and leverage the following faucets to find PMF.

Usage Metrics

1. User Engagement: Frequency of app use, session duration, and navigation paths.
2. Feature Use: Number of accesses to key features (food menus, Moodle, printing services, news and camipro).
3. Conversion Rates: From app download to active usage.
4. User Retention: Rate of returning users over time.

5. Drop-off rate: what UI transitions have high drop off

Success Criteria

1. High user engagement and retention rates - more than 70% returning users.
2. Low crash rates and quick resolution of identified issues.

User Analytics

1. Implement Google Analytics for behavioral tracking and Firebase Crashlytics for stability monitoring.
2. Anonymize data to comply with privacy standards.

A/B Testing Ideas

1. Test different navigation flows to find the most user-friendly layout.
2. Experiment with different formats of displaying food menus and Moodle content.

Design and Implementation

Frontend

Implementation framework The app will be developed in Java, optimized for Android, and will use the Model-View-ViewModel (MVVM) architecture. This approach will ensure a structured and efficient management of the app's UI and business logic.

Dynamic UI Rendering Strategy The data layer will fetch UI descriptions in a machine-readable format for each feature, such as food menus, Moodle, and printing services, etc. These descriptions will contain details about what frontend fragments that need to be rendered for each plugin.

The ViewModel will interpret these descriptions to determine the fragments to be displayed in the view of each activity. This dynamic rendering will allow for flexible UI adjustments in response to changes in service APIs or UX improvements.

Adaptability to Service API Changes The flexible UI design will be key in adapting to any future API changes. For example, if Moodle adds new sections, these changes will be incorporated in the backend's machine-readable UI description and seamlessly integrated into the app without major redevelopment.

Security and Data Management Focus The frontend will securely store refresh tokens and cache necessary app data. This approach will prioritize the security of sensitive information and the efficiency of the app's performance.

Backend Communication via Apache Thrift The app will have a thrift client built over Android's Httpcore, that can make RPC calls to our backend server. The thrift client is going to leverage Apache Thrift for data serialization and Android's AsyncTask for dispatching requests. This will ensure efficient and reliable communication between the frontend and the backend.

Authentication

Tequila OAuth2.0[1]

The integration of EPFL's Tequila Authentication system within the PocketCampus app is crucial to adoption and success of the MVP. This is primarily for two reasons:

1. EPFL services that are part of the MVP's scope are resources protected by EPFL authentication.
2. Interfacing with Tequila increases legitimacy of the application in the end users' minds. This improves user retention since trust is important for repeated engagement from end-users.

Tequila implements the OAuth2.0 workflow that enables third-party applications to authenticate and authorize EPFL users. The MVP must implement the following workflow to enable access to EPFL-protected resources. Care must be taken to avoid any potential leak of user's private information, even in the event of a data breach.

Frontend (Android Application)

1. Initiate Authentication: The Android app initiates the OAuth flow by opening a WebView or using a browser intent to navigate to the Tequila authorization URL with the necessary query parameters (`response_type=code`, `client_id`, `redirect_uri`).
2. User Login and Authorization: The user logs in using their EPFL credentials and authorizes the PocketCampus app. This step is handled entirely by Tequila's authorization server, ensuring that the user's credentials are not exposed to us.
3. Receive Authorization Code: Upon successful authorization, Tequila redirects to the specified `redirect_uri` with an authorization code. The Android app captures this redirect and extracts the authorization code. This code has a single-use validity.
4. Send Code to Backend: Instead of sending a request to Tequila and exchanging the authorization code for an access token directly from the Android app, we securely send the code to our backend server. This is crucial for keeping the `client_secret` secure.

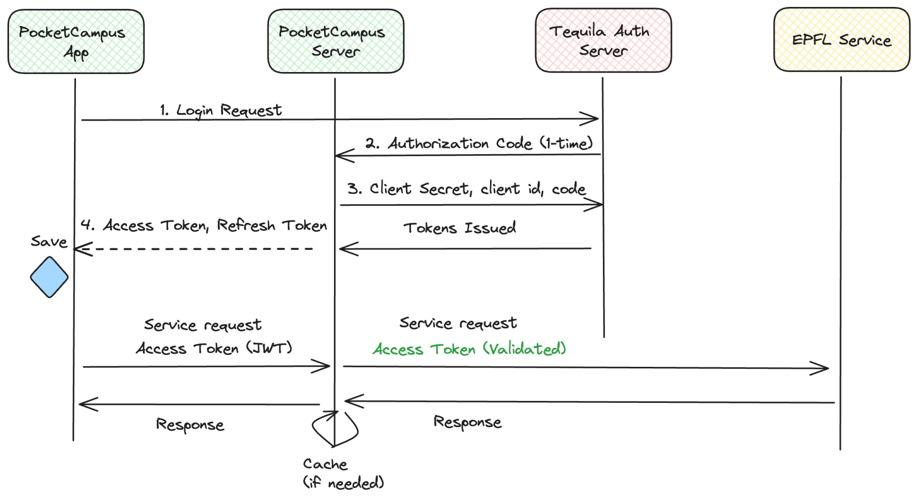
[1]Standards-based single-sign-on were not maturely deployed at EPFL a decade ago. The description is simplified here for clarity, and assumes the infrastructure available at EPFL in 2023.

Backend (Server Application)

1. Receive Authorization Code: The backend server receives the authorization code from the Android app.
2. Exchange Code for Token: The server makes a POST request to Tequila's token endpoint with the authorization code, `client_id`, `client_secret`, and `redirect_uri` to exchange it for an access token and a refresh token.
3. Receive and Store Tokens: On receiving the access and refresh token, we are careful to store neither on the server, and return them to the app client. The access token acts like a session JWT that can be used to make requests to Tequila's resource server on behalf of the user only for a short amount of time, ensuring that our backend cannot imitate the user.
4. API Requests: Our Android app makes API requests to the PocketCampus backend server, along with the access token in the request headers. The token's validity is checked on the backend (since it is a JWT), and if valid, it is used to retrieve information from any EPFL service protected by Tequila authentication.
5. Token Refresh: If the access token has expired, the validation check fails, and an INITAIATE_REFRESH response is sent back to the client app. This triggers a refresh flow where the client resends the refresh token to our server that uses the refresh token and client secret to obtain a new access token.

Security Considerations

- HTTPS: We must ensure all communications between the Android app, the server, and Tequila's servers are over HTTPS.
- Client Secret: the `client_secret` must not be exposed in the Android application. It should only be used on the server.
- Token Storage: Store access and refresh tokens securely on the client. Consider encryption and secure storage practices such as Android Keystore.
- Redirect URI Handling: Ensure that the custom URI scheme used for the redirect URI is unique to our app to prevent interception by malicious apps.



Moodle Auth

Since moodle has its own authentication system, we swap the Authorization Code obtained in step 2 for a moodle token and store it on the client device like the refresh token. This token works similar to a refresh+access token with long expiry, which is why we do not store it on the backend.

Backend

Application Logic The backend server will host all necessary application logic, enabling the client application to access and consume data from EPFL services.

It will handle communications with various EPFL services and fetch data as per user requests.

Framework The backend is developed as a monolithic Java application in Jetty, providing a stable and efficient web server environment. Moreover, we will create replicas of the Jetty container for load balancing.

Dynamic View Rendering The backend will determine changes in the frontend's rendered view based on API responses from EPFL services, like sending an additional card during course evaluations. It will implement APIs to fetch and manage the list of frontend plugins, complete with rendering metadata and associated callbacks for the Android client app.

Data Scraping Based on each service's functional requirements, cron jobs will regularly update data, ensuring the app constantly has access to the latest information, like updated food menus and new news items. Additionally, these recurrent tasks persist success/failure and other relevant statistics in the database so that we are able to track history and status across failures and app restarts.

Frontend Plugin Support For each frontend plugin, tailored backend APIs will be implemented.

For example, the API for the printing service will handle requests for the latest print & finishing options and configurations, while the Moodle plugin API will manage different kinds of filters and course data retrieval.

Responsive Backend-Frontend Interaction The backend will ensure a responsive interaction with the frontend, updating UI elements as necessary based on current data and user interactions, and act on registered notifications as needed. For example, for the camipro service, the app can run periodic queries to fetch camipro updates and reflect them directly on the frontend. Additionally, if the user enables "low balance" notification, the app can fire an RPC call to the backend that will then submit a notification to the FCM queue which the app will then receive and show to the user. In effect, app can itself trigger an API endpoint on our server to "pull"[2] a notification.

Data model

We will use MySQL for the database. Non-user-identifying information with high request volumes, such as food menus, will be cached to improve response times. In terms of user data, we are committed to privacy; thus, we will store only non-identifiable information. We document details of the schemas needed below. The exact schema will be decided when we begin the MVP implementation.

Authless Data

FoodMenu Purpose - Contains information about the daily food menus.

Fields - MenuID (Primary Key), Date, MealType, DishName, Ingredients, NutritionalInfo, Allergens, Price.

Notes - Aimed at providing quick and anonymous access to daily dining options.

News Purpose - Contains information about the news articles published by EPFL.

Fields - NewsID (Primary Key), Date, Headline, ImageURL, originURL, content

Notes - Aimed at providing access to the latest news from EPFL research.

[2]Pull notifications, popularized by Whatsapp, is a way for a client app to initiate a sendNotification to itself. This is in contrast to a push notification where the backend decides to send a notification based on some condition.

CronTab Purpose - To keep track of the status of scheduled scraping and caching tasks

Fields - UUID (Primary Key), Target = (Food/News/..), Datetime, Status, CurlRequest

Notes - Every time a cron is successful, we add new entries for the next cron. On failure, we need to investigate and restart the automated process manually.

Auth UserData

UserProfile Purpose - Manages user-specific data with a focus on privacy.

Fields - UserUUID (Primary Key, hashed value of EPFL SCIPER number), CourseList (list of enrolled courses)

Notes - The UserUUID is a hashed value ensuring that individual users cannot be directly identified. The CourseList associated with each user is stored without any personally identifying information.

NotificationTasks Purpose - Manages user's FCM device_token and

Fields - UUID (Primary Key, not linked to a user), UserUUID (Foreign key, to be able to map which (anonymous) user registers notifications), DeviceToken (FCM device token)

InMemory Data Here we list all the data that stay within the memory of an API call's execution OR the app's data layer. These are still important to standardize because they form the request-response structure that we will have to specify in Apache Thrift.

Printers. For printing service, our API receives the print & finishing options that a user specifies, along with the actual file to print before creating and submitting the job to EPFL print queue

Campiro. This is requested by the app and the response is returned to the app for display without storing transactions or balances on the backend database. We fetch and transmit transaction lists and balance.

Moodle. This is also requested by the app and the request is directly routed to Moodle APIs. Our backend only serves as an intermediary for data-model massaging in case the Moodle APIs change.

Infrastructure and Deployment

The backend server will be hosted behind an Apache web server that will serve as a reverse-proxy to either our backend server or directly to one of the EPFL websites in case needed. It will also load-balance between several backend replicas that are each running in a Jetty container.

We will host the server on a Mac mini plugged into the EPFL network. This lets us avoid connecting our server to the EPFL VPN - which will inevitably slow down the app. All printing jobs are submitted to EPFL's print queue (hosted by the myPrint service) directly from within EPFL's network.

Test Plan

Plugin Tests Since we have a dynamic frontend client that can change views, we should test frontend rendering for all plugins we can compose and support.

API Integration Tests We will need to write several tests that routinely check whether our implemented integration to EPFL APIs are working or not. This is crucial both while rolling out new changes and for monitoring changes to these services so that we can be proactive instead of reactive to failures.

Performance Testing We should stress test the app based on projected peak traffic during different times of the day/week/semester. For example, around lunch hours, when everyone is looking for the food menu, we should expect near full MAU as concurrent requests.

We should measure API latency and try to keep it sub-1s because that is the point beyond which users start to notice it.

Timeline/resource planning

Execution Roadmap

The MVP development is planned to be executed over a 14-week time period. This includes setup, development, testing, rollout, and feature iteration to find product-market fit. We envision several intermediate milestones that will help us assess our progress with a team of 4 people, 1 frontend developer, 1 backend+DevOps, 1 UI/UX Designer (part-time), and 1 software tester (part-time).

Milestone 1 Design Completion and Technical Setup

- Documentation of EPFL APIs, request-response parameters
- UI elements that can serve as building blocks for plugins: dropdowns, filters, buttons, menu bars, cards, etc.
  - This list depends on parameters and support on the APIs
- Figma mockups and user flow documents
- Tequila registration and integration
- Tests to populate UI elements with EPFL-like data
- Tests to check EPFL API status

Sprint/Week Number	Objective	Outcomes
Week 1	Kickoff and Documentation	EPFL API Investigation, Documentation and Scraping prototype scripts; set up project tools, and establish MVP scope.
Week 2	Design and Technical Setup	Initial wireframes, and user flows created, started UI mockups with UI elements for plugins. Configure Tequila Auth.
Week 3	Design Finalization & Application Skeleton	Finalized UI/UX designs, set up development environments, and backend APIs to populate simple UI elements.

Milestone 2 Core Feature Development

- With appropriate EPFL mock integration already in place, quickly prototype features each week
- Test on several devices; revisit design - aspect ratio, feel on different screen sizes and devices

Sprint/Week Number	Objective	Outcomes
Week 4	Food and News Feature Development	Developed two end-to-end features - Food menu, and news, along with basic frontend-backend integration, began unit testing.

Sprint/Week Number	Objective	Outcomes
Week 5	Printing Service Development & TestingCron Job Automation	Completed Printing Services feature, continued integration and unit testing - supporting several file types, testing on real printers across campus. Setup appropriate cron jobs to cache food and news data.
Week 6	Moodle Development & Testing	Develop Moodle Integration features, integrated all features in frontend, internal testing.
Week 7	Unit and Integration Testing	Write all remaining unit tests, check frontend-backend integration with Integration tests.

Milestone 3 Internal Testing and Pre-Launch Preparation

- Run end-to-end tests for several devices
- Do stress testing for peak load projections, avg load, and run API latency numbers
- Set up appropriate error handling and sentry logging for API failures
- Set up Google Analytics to track user events, setup analytics dashboard, and ensure correct tracking behavior.
- Firebase FCM for notifications

Sprint/Week Number	Objective	Outcomes
Week 8	Internal Testing & Analytics/Monitoring pipelines	Thorough internal testing, load testing, feedback collection, bug fixing, performance testing
Week 9	Pre-Launch Preparations	Final bug fixes and optimizations, prepared app store and launch materials.

Milestone 4 Initial Rollout and Feature Iterations for PMF

- Rollout app to alpha testers

Sprint/Week Number	**Objective**	**Outcomes**
Weeks 10-11	Initial Rollout / Alpha Testing	Deployed app to production, monitored app performance, and gathered user analytics/insights and feedback. Active hotfixes as needed.
Week 12	Feature Iteration 1	Identifying drop-off points, optimizing user experience, rollout updates, improving UI, possibly
Week 13	Beta Testing	Initiate marketing, getting users on the app, promptly addressing feedback to gain trust
Week 14	Full Rollout	Press marketing pedals, offering incentives to join the app - free meals to lucky winners,

Development Resources

The primary cost of development of the product comes from developer cost. We estimate the need to secure 4 months of cash for a 4-person team to meet planned and unplanned scenarios. Realistically, this project would need two full-time developers (one on frontend, one on backend+devops), along with two part time contractors for UI design and Testing. We expect the contractors to contribute a total of 2 person-months each, over the 4-month development + GTM timeframe. We overprovision cash for a total of 4 person-months to meet unexpected deviations from our envisioned roadmap.

Function	**Required person-months**
Frontend Android Developer	4
Backend Java Developer	4
UI/UX Designer	2
Software Tester	2
Surplus Cash Reserves	4

Deployment Resources

We will need upfront capital for the purchase of 2 MacMini so that one can serve as a redundant replica in case of hardware failures. We will have to purchase a paid plan on Sentry since we will be generating huge amounts of traffic and will need appropriate performance monitoring.

Item	**Cost / unit**	**Units**	**Totals**
MacMini	500 CHF	2	1000 CHF
UPS	500 CHF	1	500 CHF
Sentry Pro	100 CHF / month	4	400 CHF
Firebase	Free Tier	-	0
Google Analytics	Free Tier	-	0
TOTAL	1900 CHF		

Maintenance and Upkeep

We will need to constantly monitor services since unexpected changes to APIs can cause temporary downtime. Moreover, as EPFL services evolve, we will need to constantly update our database schema as well and perform relevant database migrations.

Assuming we land EPFL as a customer by the end of our 4 month timeline, the only cost of maintenance is the cost of developers.

Business model

At the core of our business model for PocketCampus is the understanding that our primary customer is the EPFL administration, rather than the students themselves. Our value proposition to the administration hinges on streamlining and centralizing access to several of EPFL's most viewed content and services. This consolidation not only enhances the campus experience for students and staff but also significantly improves administrative efficiency by providing a unified platform for information dissemination and student engagement.

The nature of PocketCampus as a comprehensive campus app requires a gradual and iterative development process, aligning with the evolving needs and use cases of the EPFL community. We believe that it could take several months to fully understand and integrate the diverse needs of such a dynamic user base into the app. This period is critical for identifying key functionalities, refining existing features, and ensuring the app's alignment with the varied aspects of campus life.

Recognizing this developmental trajectory, **we plan to propose a one-year lock-in contract with EPFL**. This duration is strategically chosen to allow sufficient time for the app to mature and demonstrate its full potential. Unlike applications with immediate mass appeal, such as games that can go viral overnight, PocketCampus is expected to gain traction and user loyalty progressively. Over the course of a year, we will be able to collaboratively work with the administration and users to tailor the app, ensuring it meets and exceeds the expectations of the EPFL community. This approach not only ensures a well-rounded and effective product but also fosters a strong, ongoing partnership with EPFL.

The one-year lock-in period also provides a stable financial and operational framework for us to invest in the necessary resources, development, and support required to make PocketCampus a success. It offers a predictable revenue stream and a clear timeline for both development and return on investment, benefiting both PocketCampus and EPFL. Additionally, to incentivize EPFL to promote our app, we plan to offer attractive rates if the contract is made based on the total EPFL population instead of active users. This is aimed at creating an operational synergy that is beneficial to both EPFL and PocketCampus in the long run.