Puzzle Time
Team 11

Team Members and email addresses:
Eric Smith       e380s105@ku.edu
Isaac Gonzalez   i482g724@ku.edu
David Krall      d277k748@ku.edu
Brian Kennedy    b919k168@ku.edu
Patrick Shields  pss@ku.edu

Team Meeting time:
  12:00-2:00 PM Mondays
  1:30-3:30 PM Wednesdays

Lab Meeting time:
  2:15 PM Monday

Project Description
Puzzle Time is a productivity application for mobile devices that incorporates “gamification” into the traditional time management software. Using the device’s internal timer, the users will be able to keep track of time spent working and time spent on breaks to increase their productivity. The “gamification” is applied through simple puzzle games during the breaks. A social aspect is included by the exchange of small trophies in the form of puzzle pieces between users to incentivize continued use. Similar applications are common on desktops and without a gamification aspect, however our app plans on using the mobile platform to provide a new product. By the end of the class we hope to have at least a minimal working product which contains the core application and a few games.

It is our hope that by promoting community involvement, we can encourage mutual accountability in our user base and achieve higher productivity as a result. To the same end, the app will have the optional setting to mute other notifications, limiting interruptions. The timer will use common business timing practices to intersperse work with breaks, making Puzzle Time an ideal workplace application.

Project Milestones
1st Semester:
  Project Proposal Report (October 24th)
  Design Documentation (October 31st)
  Code Outline (December 9th)
2nd Semester:
  Backend Completed (February 22nd)
  Frontend Completed (March 1st)
  Game Interface (March 8th)
  One Game Completed (March 15th)
  Android/iOS Deployment (March 29th)
  Additional Game Development (April 19th)
Project Budget

Apple App Store developer fee ($100)
Google Play Store developer fee ($25)

Work Plan

David:
  Game UI
  Game Development
  Documentation
  Deliverables

Patrick:
  Game UI
  Timers and Notifications
  Game Development
  Android/iOS UI
  Documentation
  Deliverables

Eric:
  Server and Database
  Game Development
  Documentation
  Deliverables

Isaac:
  Game UI
  Server and Database
  Game Development
  Documentation
  Deliverables

Brian:
  Game UI
  Timers and Notifications
  Game Development
  Documentation
  Deliverables

Github link
  https://github.com/ersmi/puzzle_time

Preliminary Project Design

Generalization of the Project

At the highest level, the software works through communication from distinct frontend and backend applications. The frontend application is the mobile application that hosts the games and interface for the user to interact with. The backend application consists of a database and server to process user requests and store the progress and data the user
creates through the frontend. The backend also serves as the database for allowing users to connect and interact with each other in the frontend application.

![Use Case Diagram for Puzzle Time](image)

**Figure 1) Use Case Diagram for Puzzle Time**

**Frontend Overview**

The front end of Puzzle Time will be the bridge between our users and the application to allow for a user-friendly interaction. In order to accomplish the goals of our project, we must take careful consideration about the choices we make in our implementation of the front end. In this section, we'll be discussing details on how our frontend will be implemented in three main topics: Platforms & Frameworks, Interfaces, and Functions.

**Platforms & Frameworks**

In order to help us develop Puzzle Time’s front end we are going to use Ionic, an open source HTML SDK that uses HTML, CSS, and JavaScript to help build mobile apps. From the get go, we decided we wanted to release our mobile app on the Android and iOS platforms and with Ionic, it is able to render web pages easily for Android and iOS. This will keep us from spending more time translating our front end implementations between platforms. Because Ionic uses HTML, CSS, and JavaScript we’ll get the advantages of keeping our front end development under a single umbrella while still allowing for freedom in our designs.
Under Ionic, we will be using Angular JS, a framework designed to make the deployment process easier and support the infrastructure of the application. Like Ionic, Angular operates cross platform, which will fulfill our need to run on multiple platforms. Ionic has been built and advertised to run on top of Angular, resulting in an all-around SDK to successfully support and develop our app.

Screens/Interfaces

The frontend is mainly interfaced through several screens programmed using JavaScript and HTML within the mobile application. The program will first direct users to login, and then prompt to start the timer. The timer will count down from some amount of time and afterwards prompt the user to select a game and reward. The games will use a JavaScript frame within the application to avoid needless complexity and connections to outside websites and applications.

The application will also have interfaces to view information and statistics about its use, user credentials, and view information about friends they've added within the application. All of the interfaces will be linked using standard mobile app menus to create an intuitive interface for the user. The interfacing with other users of the application will help promote use of the app through the social networking aspect.

![Figure 2a) Main Menu](image1)
![Figure 2b) Work Timer](image2)

Functions/Features/Role in Product

From a frontend view, the product’s primary features will be accessible from the main menu. The app’s menus will be streamlined and fairly standard, which is made possible by the Ionic framework. This will make the app intuitive and accessible. When using the product’s primary functionality, a display featuring the amount of time remaining in their study period will be shown onscreen. As a secondary functionality, the app will cycle among the puzzles that have been earned by other users, displaying them alongside the remaining time during the work period.
During standard use, the main attraction of the app is the use of games to motivate productivity. These games are presented within the framework of the app and may be chosen from a scrollable menu when the timed work period ends. Games will be programmed using JavaScript. The inclusion of these games provides uniqueness and generates appeal, in addition to their more obvious role as a motivator.

The app’s frontend also features social functionality, which manifests in several ways. First, the user may choose to obtain a puzzle piece that a friend has already earned. The puzzle in question will be chosen from among the friend’s puzzles-in-progress and completed puzzles. These will be displayed by the GUI, with a page assigned to each and buttons on either side to allow easy transition from one to another. Users may also view information about other users who have been registered as friends. The Friends List can be accessed from the main menu, and it allows the user to view puzzle pieces that these friends have earned as well as view their respective Friends Lists. The Friends List functionality allows a user to view a thumbnail list containing links to the profiles of their registered friends.

A final feature of the app is the process of collecting puzzle pieces. Puzzle pieces may be earned by participating in the time-based efficiency management system that puzzle time enforces. Puzzle pieces can also be viewed from a user’s profile, with each puzzle assigned a single page and direction buttons placed on either side of the page to facilitate easy access. To display the pieces obtained by either the user or a friend, Puzzle Time will “mask out” all but the obtained parts of the puzzle after dividing the source image into a number of fragments of equal size. The user’s growing collection of puzzle pieces plays a largely motivational role in the app.

Backend Overview

The purpose of the backend within the Puzzle Time application is to host the database for the mobile application to use for data storage. This is accomplished through the use of a web server that uses HTTP requests to communicate with the mobile application in real-time.
Platform and Language Choices

The platform for the backend is a Linux hosted web server with an outward API to supply the application with the necessary database and server functionalities that it needs to properly supply user interactivity. The frontend runs on a variety of devices, so the server uses a generic web application design to provide a level of common functionality towards each of the devices through the common HTTP interface. This allows for the server to process queries from both iOS and Android systems without the need for considering the OS of the source.

For a language we have decided to use Python for the server side backend programming. The choices was decided upon for the purpose that it would be easy to promote quick development and the fact that there are plenty of frameworks that provide simple web APIs. The trade-off for the ease of development is the costliness of the Python platform. This is mitigated by having the server only serve an API, since the frontend will handle most of the actual interactivity with the games and therefore will be able to operate with relatively low memory costs. This will allow the normally large performance decrease when compared to other languages to not have a meaningful impact on the backend. We will be using the Django framework for interfacing Python with the web. This will allow for the use of method and MVC conventions that makes for easy development and deployment of the HTTP server.

The backend is designed and uses Postgres as its database software. For testing purposes, the backend uses SQLite in the instance that Postgres isn’t properly installed on the system. This allows for anyone with a proper python installation to run a test server without the need to install linux system packages for Postgres. In deployment, we use the Postgres database software with the Heroku Postgres service.

Interfaces

The backend's purpose within the project is to mainly serve as an access point for the frontend to use for databases, concurrency, and persistent sessions. Being a web server, most of the interfacing will come from the use of HTTP requests in order to tell the web server to return specific sets of data. These requests can be sent within the application using the Angular JavaScript framework to craft the requests for the server. The requests are separated into categories for each of the relevant tables in the database, those being puzzles, pictures, and users. Each category has support for GET, POST, PUT, and DELETE requests, which will give the frontend complete control over the table entries. The server can also be used to present the API through a web browser through traditional web URLs. This allows for easy debug and use in environments outside of the mobile application without losing the secure authentication.

Actual management of the server is done through the use of a CLI interface supplied by the Django application. This allows for remote manipulation of the database and server
functionality through SSH or a similar remote machine service. It will also allow for scheduled commands and queries to run on the database periodically to perform tasks such as maintenance. Django also has a built-in GUI for this interface that supplies a small subset of its common features through a web browser.

The backend also serves as an interface to manage the authentication and credentials of user accounts and sessions. This allows for persistence between logins and devices, should the user have multiple devices that they wish to use the application with. The backend also uses the authentication to supply users with the correct details and settings that they have previously selected to apply to the frontend application. The backend uses the native Django password and email encryption in order to maintain a secure database for the users.

API Implementation and Communication with Frontend

The backend supplies the application and users with information regarding the status of the games, timers, and user accounts through the API implementation. As stated before, these are available to the application through HTTP requests. The API supports the release of the user information, puzzles being worked on, and pictures to the frontend. The API is restricted through authentication to make sure the user attempting to access data can only see what they have permission to. This enables us to maintain a reasonable amount of privacy for users in regards to their posted content.

Database Design Schema

The database is formatted into the following tables:
- User Credentials and Information
- Puzzles
- Pictures
The user credentials and information table is responsible for holding the user login information as well as their progress within the application’s games and puzzle pieces obtained. The user table holds information about what other users the user in question has access to. The table also contains links to each of pictures and puzzles created and uploaded by the user.

The puzzles table holds progress of each user’s puzzles. This will be done by using links to the user id in the credentials table and the picture file from the picture table. Each puzzle also contains a link to the picture table entry associated with it as well as the current progress stored as a simple binary string.

The pictures table holds the pictures uploaded by users. Each picture contains a link to the user that has uploaded it as well as the puzzle ids for the puzzles that are currently using the picture.

Hardware Implementation Details

For deployment we have decided to use Heroku’s web hosting service. We made this decision based on the support Heroku has for Django applications as well as its own Postgres services. Heroku also offers a free tier of web hosting, as well as a simple git based deployment system that allows for iterative development easily.

Ethical and Intellectual Property Issues

For our program, when a user uploads a photo to the application, they will have to possess and surrender some intellectual property rights for the image. In order for the app to be used, we would have to take these images and reproduce, distribute, and publicly display them to others. Depending on the implementation, images uploaded by the user publicly may be visible to an outward API. Therefore to ensure the application is allowed to use the images, the user will have to acknowledge these rights that they are giving away. Should the user not have rights to the original image to begin with, this also leads to ethical issues when enabling the user to share the image illegally. Another ethical consideration involving the images is the potential for explicit images to be uploaded by the users. This would expose users to violent or inappropriate images uploaded by other users. The overall reputation of Puzzle Time would also be negatively impacted by the presence of these photos given the perceived target audience for the application.

There are two options that we could take when it comes to copyrighted images being uploaded. We could have a policy saying that you shouldn’t upload copyrighted materials, then we could either not enforce this policy or attempt to enforce it. If we analyze this in terms of monetary costs, the cheapest option, in terms of time and money spent, would be not to deal with uploaded content at all. Unless we actually obtain a large user base and popularity, the likelihood of legal threats due to copyrighted content seems minimal. The other option is policing images. If we did obtain a large enough user base where this would be necessary, we could have user moderation to offset any cost. It is doubtful that the images would fall under fair use, even if they were low resolution.
Our application will give users the ability to upload personal information about themselves. We may have certain obligations to our users to afford them a certain level of privacy, and certain ways that we should or should not use this information. How much security are we obligated to offer? What are acceptable ways of using user data? If we could use this data to somehow enhance services for our users, that would be a helpful and mostly harmless usage. If our application was ad supported, we could use personal information to target ads. Analyzing this in terms of “Respect-for-Persons”, we would ask “What if everyone did the same thing?” If we’re talking about using user data to enhance our services, such as having location data for photographs, or some other metadata, this would probably be a net benefit for our users. The important thing to ensure is that our users know how their data is being used for these services, and to give them the ability to opt-out. We should also ask “Would I be OK with people using my data in this way?”, but again it comes back to getting users consent in however we use their information.

Change Log

Updated meeting/lab times for second semester.
Updated project milestones
Updated project gantt

Design

Frontend

Backend
  Updated deployment design from self-hosted to Heroku.
  Updated database schema with table links.
  Changed database software from SQLite to Postgres
  Added additional detail to frontend/backend interfaces.
  Added distinction between testing/deployment environments.
  Added information about Django admin GUI