BucStop

Vision Statement— A website built by ETSU Students for ETSU Students to upload games they have written so other ETSU Students can play them between classes.

Objectives —
The website will feature a collection of classic arcade style games written by ETSU students of varying skills and interests. Each game should ideally be able to be played to completion in 20 minutes or less with a scoring system that allows the player to register their initials if they get a high score.

The website should have a way for ETSU students to submit new games or updates to existing games. The games must be compliant to ETSU standards of conduct. Each game should be written to be fully loaded into a browser frame so that play is not dependent on continuous network connections (since ETSU wifi is notorious for dropping connections). Leaderboards and site statistics can show which games are the most popular, which are the newest, etc. which students can then leverage in their resumes.

There are features associated with this solution that should be of interest to all computing students; Cyber Security, Information Technology, Information Systems, and Computer Science.

Cyber students can manage access to only students that are accessing the website via ETSU wifi or network systems (CIDR blocks and AWS Security Groups). They can leverage mac addresses as well to track individuals without requiring logins. They can monitor and do analytics on unauthorized attempts at access. They can analyze submitted games for potentially malicious code. While several of these actions could be done manually, it is the goal to automate as much of these actions as possible. For example, an automated alert (SMS text) could be sent to specific persons if unauthorized attempts exceed a certain threshold over a period of time (similar to a Denial of Service attack).

IT students will be able to put their system administration and web knowledge into practice by supporting DevOps and automated deployment, logging, system restoration, and rollbacks. The solution employs containerization in support of emergent architectures (IT students need to work with Cyber students to ensure each container is properly configured to prevent any backdoor access). Ideally, each release of the solution will be deployed to a testing environment where automated and manual tests are performed to verify and validate the solution. Once passed, the solution would then (automatically?) deploy to a staging environment where load, stress, and security testing (among other “ility” tests) are performed. Once passed in staging, the solution would then (automatically) deploy to production in a way that does not impact active users of the website.

IS students can perform a number of data analytics and develop workflows for the website. It would be a nice addition to a student resume if they wrote the most popular game for a semester. IS students can define what data elements need to be collected and how so that this claim can be made in a valid way. For example, is it just the number of people that viewed a game on the website, or the number that selected the game to play, or the number of people that played the game to completion, or some combination of all of these? Is the number of plays or the number of people that played (since we don't require a login, how would you figure it out? Hint: Mac Address). There needs to be a workflow defined for submission of new games and for updates to an existing game. Since students graduate (eventually), is it permissible for other students to update a game written by someone else or do they have to submit it as their own game? Does the original author of the game have any say if they are still around (as a student, staff, or faculty)? How is versioning of the games done and how does that impact the data collection and visualization? While the IT students leverage logging for keeping an eye on the health of the solution, the IS students can leverage logging to collect statistics for data analytics.

Computer Science students will likely enjoy designing and building the games as well as expanding the features of the website itself. They will have to balance upload speeds, performance/response times, with engaging game dynamics and graphics. They can take advantage of local storage or cookies, etc. but should be careful not to require either for the game to work at some basic level (e.g. they could use local storage to remember the player's initials so they can use it as the default when the player completes another game play). Automation of testing is important to ensure that new versions of the games don't break existing features. CS students can leverage logging for debugging and troubleshooting.

Current Status –As of the Spring 2025 semester conclusion, the solution has three games (Pong, Tetris, and Snake). Each game is deployed in its own microservice within its own Docker container. The main website communicates with these microservices via an API Gateway which is also in its own Docker container. The main website is also in its own Docker container. Deployment is semi-automated using Docker compose and Docker deploy.

All the components are in a single repository in GitHub. Having multiple deployable components in the same repo is often referred to as a poly-repo.

The AWS security groups were investigated but not implemented. Currently, players have to enter an email address to see the games. A simple regex is applied to the email address to see if it is in the etsu.edu domain. This should be removed and replaced with CIDR blocks and security groups at some point.

The last team implemented logging and recovery files. The logs can be used to verify everything is booted properly and connected. Heartbeat log entries are generated but I'm not sure if these work as intended. Recovery files were a way to restore the persisted data (e.g. as part of a rollback). However, I don't think versioning was fully represented in the process (i.e. each recovery file should be associated with a version of the solution so that you can't recover persisted data that isn't compatible with the running version of the solution). This is where the power of semantic versioning would be put into play.

The solution is located at https://github.com/etsuDummy/Spring25_BucStop

Scope - Each team in each new semester can choose what they want to focus on adding to and expanding on for the solution. In the past semesters I have selected the team that demonstrates the best Scrum/Agile practices and adherence to the SDLC as the baseline for the next semester. If students were to do a Scrum of Scrum, using a single code base for all the teams, then the all the contributions of every team would advance to the next semester.

1. **User roles**—
    - Player: anyone connecting to the website via the ETSU network can play the games available at that time.
    - Game Builder: an ETSU student that designs and builds a game that adheres to the submission criteria for the website.

2. **High Level Requirements**:

Security…

- Access to the website must be limited to connections from ETSU networks
- Invalid attempts to access the site should redirect to a static text page that encourages the user to enroll in ETSU's computing department.
- Once a user has successfully gains access with a device, their mac id should be stored. Future visits from that device will be allowed even if not from an ETSU network for up to one year from the last time they used an ETSU network connection to visit the site.
- Games can take advantage of local storage and/or cookies to enhance player experience. Use of local storage and/or cookies should only be for optional aspects of the game (e.g. they could use local storage to remember the player's initials so they can use it as the default when the player completes another game play)
- All communications between the game and the website will be done in open text using RESTful API calls and stateless transactions.
- Games must be agnostic of personal identifying information.

Players…

- shall be identified by the mac id of the device they connect with.
- Shall be presented with a list of available games upon connecting to the site
- Shall be able to activate any available game to play
- Shall be able to exit any game at anytime during play
- Shall be able to register their initials (or any 3 letters they wish) upon playing a game to completion if they have a top ten score
- Shall be able to view information about any available game including but not limited to: an image of the game screen, a description of the game, instructions on how to play the game, any available release notes, the current version of the game, and a leaderboard for that game

- Shall be able to view the number of times a game has been played
- Shall be able to view the top 10 most played games

Games…

- Shall comply with the submission criteria as defined at the time the game is submitted
- Shall support all the defined RESTful APIs required to communicate with the website
- Shall be playable within a browser even if the network connection is lost after the game has started (e.g. should contain retry logic for any required calls back to the server)
- Shall have a scoring system that results in a “score” at the completion of the game play.
- Shall solicit 3 letters from players whose score is in the top ten at the time the game play ended.
- Shall update the website whenever the player exits the game before game play ends.
- Shall update the website whenever the player plays the game to completion
- Shall update the website whenever the player exits the game without starting a game
- Shall be submitted for consideration and evaluation before being made available to players
- Submissions shall include author’s name, contact information, game name, description, instructions, image, code for embedded browser play (e.g. JavaScript), category type (e.g. strategy, puzzle, etc.)
- Shall support semantic versioning when updates to the game are made
- Updates shall include release notes that include but are not limited to a list of reported defects/issues resolved in that update

Website…

- Shall present available games in a carousel style view using images for each game
- Can present a top ten most popular games view including the game name, image, author, and
- Shall track access attempts
- Shall track game play (e.g. how many unique mac ids visit the site, how many view game info for each game, how many play the game to completion, etc.)
- Shall display game play statistics to visitors of the website (e.g. most visited games, games played to completion the most, daily play trends, hourly play trends, etc.)
- Shall allow players to submit trouble tickets for a game or the website
- Shall provide a mechanism for new games to be submitted for inclusion in the site (may be automated in a manner that validates the submissions adherence to required game criteria and then spins a new microservice to add the game to the site so that no manual steps are required for the game to become available to play)
- Shall provide a mechanism for updates to existing games to be submitted

Non-functional Requirements:

- The solution shall be deployed in a way that future classes can maintain

- The code base and associated documentation shall be stored in a way that future classes can maintain and extend
- Shall have the ability to rollback to a previous version if a new release fails to work properly
- Shall support versioning of each game

3. **Any special issues or considerations:**

- Must be a client-server architecture.
- Client interface is browser based (vendor neutral).
- Client interface is internet connected device neutral (e.g. works on phones, tablets, laptops, etc.)

Things to consider for the project in the Fall of 2025

- (CY)Fix security groups for production environment (see https://github.com/etsuDummy/Spring25_BucStop/blob/main/Documentation/AWS%20Security%20Groups%20-%20BucStop.pdf) (I think you could remove outbound groups entirely and that might fix the issue; inbound restrictions should be enough). Remove the hokey login and any concept of an account. Players just play.
- (CS) Ensure the webapp doesn't have specific knowledge of what games are available through the gw (this allows you to add or remove games without negatively impacting the webapp) (see https://github.com/etsuDummy/Spring25_BucStop/blob/main/Documentation/Emptying%20_games.json_.pdf for some hints) (also see https://github.com/etsuDummy/Spring25_BucStop/blob/main/Documentation/Cooked%20API%20Gateway%20Document.pdf though I think this document is out-of-date but you should verify it; if so, delete it. See https://github.com/etsuDummy/Spring25_BucStop/blob/main/Documentation/StaticFileHosting.md for why I think the previous document is obsolete.) GatewayController.cs has hardcoded array for the 3 current games ("var gameKeys = new[] { "Snake", "Tetris", "Pong" };" It should be config file driven or discoverable, not hard coded.
- (CS) Add some games (it's not as easy as you might think)
- (IT/CS) Update API calls to support a leaderboard for each game (this should be done generically and I can walk you through some of the key requirements if you take this on)
- (IS/IT) Expand on the DevOps automation by 1) adding automated regression tests 2) include an automated smoke test to validate deployment 3) add a testing environment (where regression testing is done) and staging environment (where non-functional testing can be done (e.g. rollbacks, security groups) 4) expand logging and real time metrics and measures (e.g. boot time, tracking mach ids and playtime, time of day when site is accessed, etc.)
- (IS/CS) expand Player count to also capture when a game is played to conclusion (vs just started so that we can see how many people start a game vs how many play it to the

end) and make sure if a user plays the same game multiple times that player count is incremented with each play.

- (CS) Create a way to dynamically/programmatically add a new game to the solution and have the API gw automatically pick it up so that it appears in the webapp on the next refresh

We will do a scrum of scrums across both sections. This impacts how source control is done (see https://github.com/etsuDummy/Spring25_BucStop/blob/main/Documentation/ci-pipeline.md) , requires the POs for each team to collaborate and plan together, promotes vertical slicing (each of the suggested changes above can be done by any team but communication across all teams is needed to ensure things don't break other teams; for example, the api calls needed for leaderboards can be implemented for all existing games by one team but any team building new games needs to know about them and there should be a plan for how to migrate games to the new apis).

4. Glossary of Terms:

- **Web application** - Application software that runs on a web server.
- **Client server model** - User (client) is provided services through an off-site server.
- **Client interface** - The mechanism by which a user interacts with an application, service or system.
- **Browser based application** - An application that runs within a Web browser.
-